II

Natural Sciences Tripos Part II

MATERIALS SCIENCE

Computing Course: LabVIEW

Name..... College.....

Dr D. Bosworth

Michaelmas Term 2015-16

Athena SWAN Bronze Award

UNIVERSITY OF CAMBRIDGE
Department of Materials Science and Metallurgy

Computerised Data Acquisition in LabVIEW

Dr D. Bosworth – October 2015

This course is adapted from "LabVIEW in 6 hours" published by National Instruments and a range of NI white papers. Where relevant links have been included to provide more detail.

Introduction

LabVIEW is one of several programming languages that can be used to automate scientific experiments. It is widely used in the materials science department because it can be used to write programs quickly that have easy to use graphical interfaces. This course will cover the basics of how to use labview and then provide a chance for you to put what you have learnt in to practice.

Course Overview

The course consists of 6 hours of introductory lectures and 12 hours of practical work. We expect to you to have widely different programming experience please do not worry if you find that some tasks take you longer than other members of the group.

There are a total of 3 assessed exercises, which you should aim to complete through the term.

The course is assessed out of 30 marks. You will work ***individually*** for the first two assessed exercises and as a pair for the final one. The final report must be entirely your own work.

A maximum of 5 marks are awarded for each of the first two exercises with 10 marks available for the final exercise. You will be marked on how well your final programs fulfils the specifications, the quality of the solution you have arrived at and how well you have used the various aspects of labview to complete the problem.

10 marks will be awarded for a short report on the three programs (no more than 8 A4 pages). This should explain how your programs work, why you made your design choices and assess how well you think your final effort fulfilled your aims.

The deadline for the submission of the report will be given to you at the start of the term. A paper copy should be submitted to the teaching office. You should also combine the VIS from your project into a zip file and email this to teaching-office@msm.cam.ac.uk.

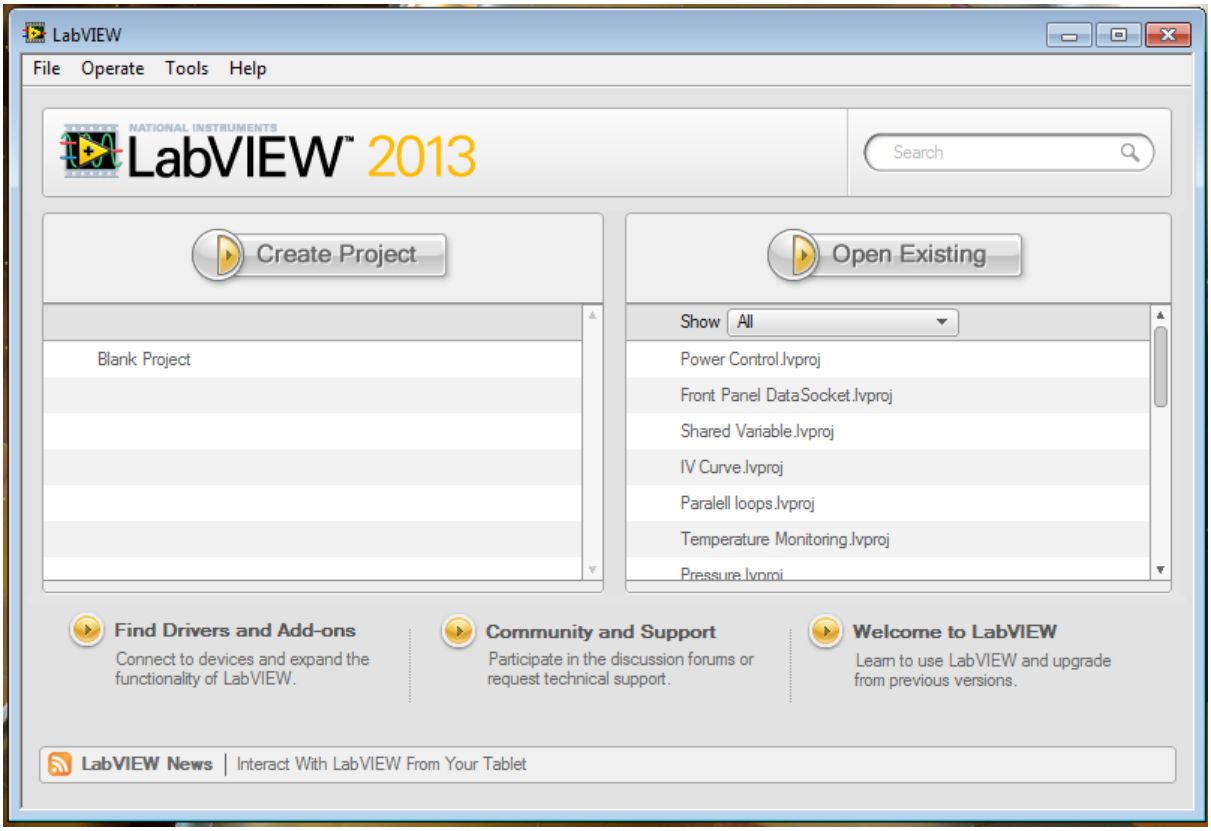
You should find you have your own computer in the computer lab and a microphone is provided, this will serve as a source of data. Work through the exercises in this booklet in order.

LabVIEW

LabVIEW opens as an application like any other. When you open the program choose to create a New blank VI. Virtual Instrument or “VI” is the term that National Instruments applies to an individual program or subroutine.

When your new program opens there are two windows. The grey coloured window is the “Front Panel” from where you interact with the program. The white coloured window is the program where you draw your program.

Labview Vis can be saved to disk just like any other document. Please don’t forget to save a copy of your work to a USB stick or other safe place so you don’t lose it. Saving different versions can be useful in case you decide you want to revert to an earlier version of your programme.



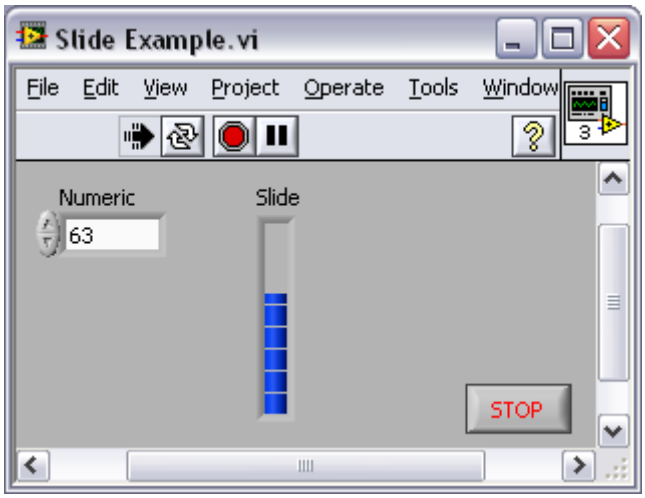
LabVIEW Basics

LabVIEW programs are called virtual instruments (VIs).

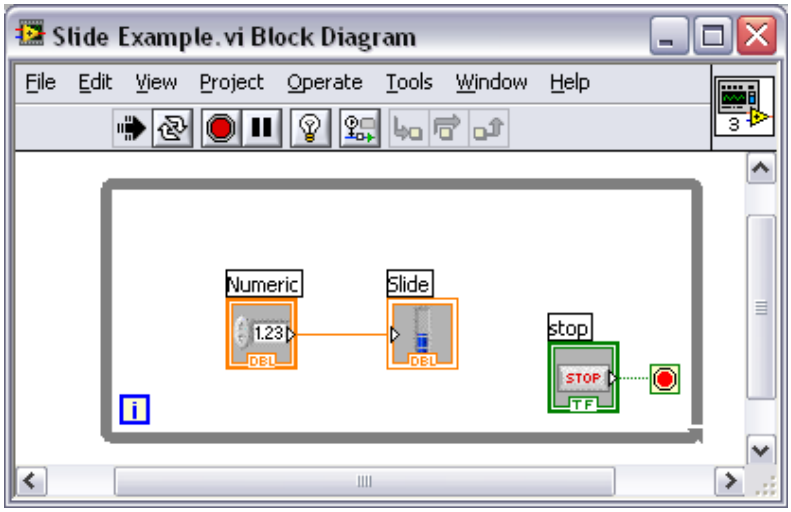
Controls are inputs and indicators are outputs.

Each VI contains two main parts:

Front Panel – How the user interacts with the VI.



Block Diagram – The code that controls the program.

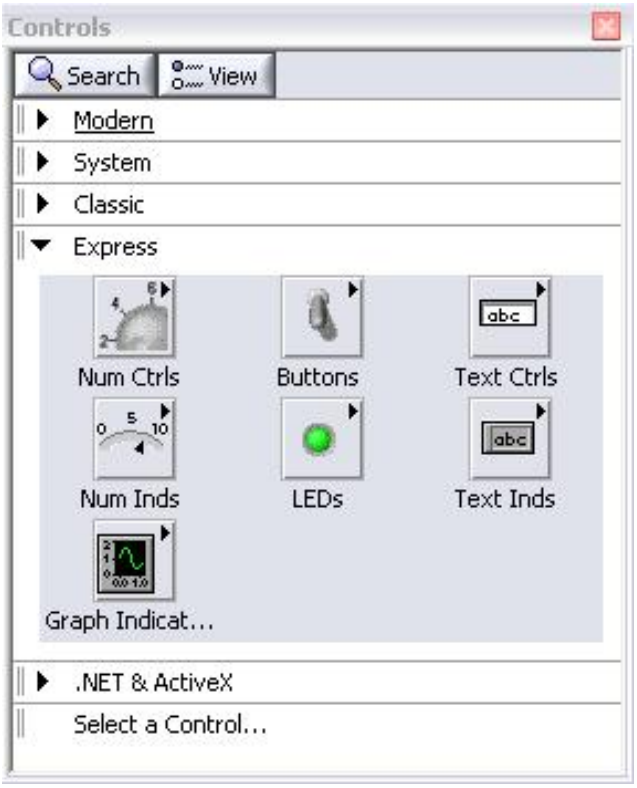


In LabVIEW, you build a user interface by using a set of tools and objects. The user interface is known as the front panel. You then add code using graphical representations of functions to control the front panel objects. The block diagram contains this code. In some ways, the block diagram resembles a flowchart.

Users interact with the Front Panel when the program is running. Users can control the program, change inputs, and see data updated in real time. Controls are used for inputs such as, adjusting a slide control to set an alarm value, turning a switch on or off, or to stop a program. Indicators are used as outputs. Thermometers, lights, and other indicators display output values from the program. These may include data, program states, and other information.

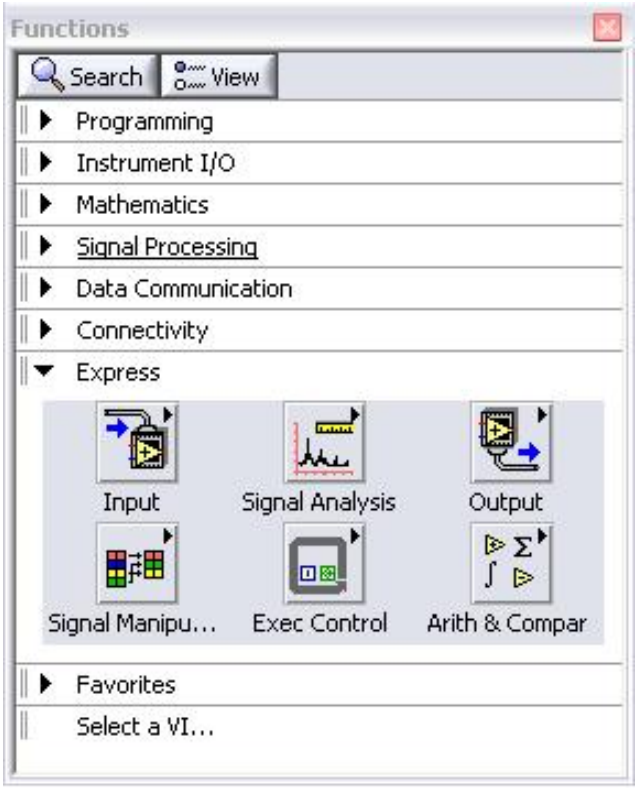
Every front panel control or indicator has a corresponding terminal on the block diagram. When a VI is run, values from controls flow through the block diagram, where they are used in the functions on the diagram, and the results are passed into other functions or indicators through wires.

Use the Controls palette to place controls and indicators on the front panel. The Controls palette is available only on the front panel. To view the palette, select Window»Show Controls Palette. You also can display the Controls palette by right-clicking an open area on the front panel. Tack down the Controls palette by clicking the pushpin on the top left corner of the palette.



1 The Controls Palette

Use the **Functions** palette to build the block diagram. The **Functions** palette is available only on the block diagram. To view the palette, select **Window»Show Functions Palette**. You also can display the **Functions** palette by right-clicking an open area on the block diagram. Tack down the **Functions** palette by clicking the pushpin on the top left corner of the palette.



2 The Functions Palette

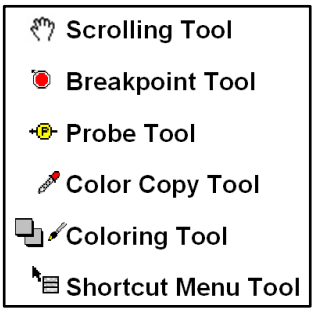
If automatic tool selection is enabled and you move the cursor over objects on the front panel or block diagram, LabVIEW automatically selects the corresponding tool from the **Tools** palette. Toggle automatic tool selection by clicking the **Automatic Tool Selection** button in the **Tools** palette.

3 The Tools Palette

Use the **Operating tool** to change the values of a control or select the text within a control.

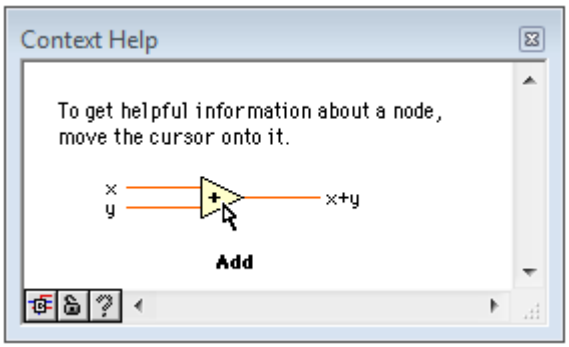
Use the **Positioning tool** to select, move, or resize objects. The Positioning tool changes shape when it moves over a corner of a resizable object.

Use the **Labeling tool** to edit text and create free labels. The Labeling tool changes to a cursor when you create free labels.



Use the **Wiring tool** to wire objects together on the block diagram.

The **Context Help** function is a quick way of finding out more about a sub-VI. Pressing Ctrl+H will bring up the window, when you hover over a VI it will display the information about it.



Objects with context help information include VIs, functions, constants, structures, palettes, properties, methods, events, and dialog box components.

To display the Context Help window, select **Help»Show Context Help**, press the <Ctrl+H> keys, or press the **Show Context Help Window** button in the toolbar

Connections displayed in Context Help:

Required – bold
Recommended – normal
Optional – dimmed

Additional Help

VI, Function, & How-To Help is also available.

Help» VI, Function, & How-To Help

Right-click the VI icon and choose **Help**, or

Choose "**Detailed Help**." on the context help window.

LabVIEW Help – reference style help

Help»Search the LabVIEW Help…

- Keystroke Shortcuts
  - <Ctrl+H> – Activate/Deactivate Context Help Window
  - <Ctrl+B> – Remove Broken Wires From Block Diagram
  - <Ctrl+E> – Toggle Between Front Panel and Block Diagram
  - <Ctrl+Z> – Undo (Also in Edit Menu)
- **Tools»Options…** – Set Preferences in LabVIEW
- VI Properties–Configure VI Appearance, Documentation, etc.

4 Keyboard Shortcuts

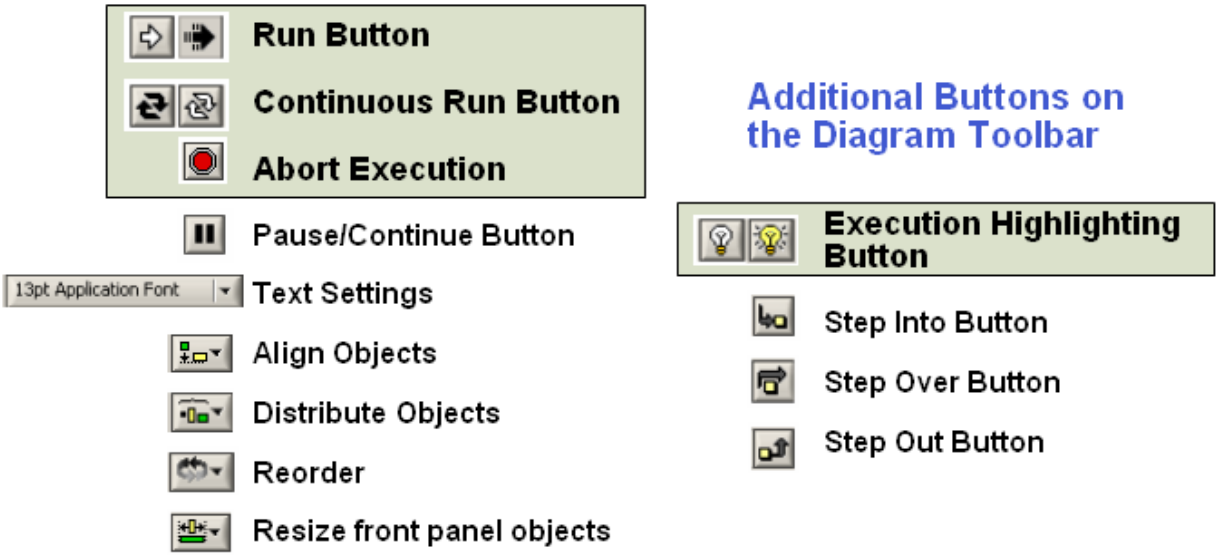
LabVIEW has many keystroke shortcuts that make working easier. The most common shortcuts are listed above.

While the Automatic Selection Tool is great for choosing the tool you would like to use in LabVIEW, there are sometimes cases when you want manual control. Once the Automatic Selection Tool is turned off, use the Tab key to toggle between the four most common tools (Operate Value, Position/Size/Select, Edit Text, Set Color on Front Panel and Operate Value, Position/Size/Select, Edit Text, Connect Wire on Block Diagram). Once you are finished with the tool you choose, you can press <Shift+Tab> to turn the Automatic Selection Tool back on.

In the **Tools»Options…** dialog, there are many configurable options for customizing your Front Panel, Block Diagram, Colors, Printing, and much more.

Similar to the LabVIEW Options, you can configure VI specific properties by going to **File»VI Properties…** There you can document the VI, change the appearance of the window, and customize it in several other ways.

There are several buttons available across the top of the program window:



5 Buttons avaliable in a LabVIEW window

Click the **Run** button to run the VI. While the VI runs, the **Run** button appears with a black arrow if the VI is a top-level VI, meaning it has no callers and therefore is not a subVI.

Click the **Continuous Run** button to run the VI until you abort or pause it. You also can click the button again to disable continuous running.

While the VI runs, the **Abort Execution** button appears. Click this button to stop the VI immediately.

Note: Avoid using the **Abort Execution** button to stop a VI. Either let the VI complete its data flow or design a method to stop the VI programmatically. By doing so, the VI is at a known state. For example, place a button on the front panel that stops the VI when you click it.

Click the **Pause** button to pause a running VI. When you click the **Pause** button, LabVIEW highlights on the block diagram the location where you paused execution. Click the **Pause** button again to continue running the VI.

Select the **Text Settings** pull-down menu to change the font settings for the VI, including size, style, and color.

Select the **Align Objects** pull-down menu to align objects along axes, including vertical, top edge, left, and so on.

Select the **Distribute Objects** pull-down menu to space objects evenly, including gaps, compression, and so on.

Select the **Resize Objects** pull-down menu to change the width and height of front panel objects.

Select the **Reorder** pull-down menu when you have objects that overlap each other and you want to define which one is in front or back of another. Select one of the objects with the Positioning tool and then select from **Move Forward**, **Move Backward**, **Move To Front**, and **Move To Back**.

Note: The following items only appear on the block diagram toolbar.

Click the **Highlight Execution** button to see the flow of data through the block diagram. Click the button again to disable execution highlighting.

Click **Retain Wire Values** button to save the wire values at each point in the flow of execution so that when you place a probe on a wire, you can immediately obtain the most recent value of the data that passed through the wire.

Click the **Step Into** button to single-step into a loop, subVI, and so on. Single-stepping through a VI steps through the VI node to node. Each node blinks to denote when it is ready to execute. By stepping into the node, you are ready to single-step inside the node.

Click the **Step Over** button to step over a loop, subVI, and so on. By stepping over the node, you execute the node without single-stepping through the node.

Click the **Step Out** button to step out of a loop, subVI, and so on. By stepping out of a node, you complete single-stepping through the node and go to the next node.

A simple VI

When you create an object on the Front Panel, a terminal will be created on the Block Diagram. These terminals give you access to the Front Panel objects from the Block Diagram code.

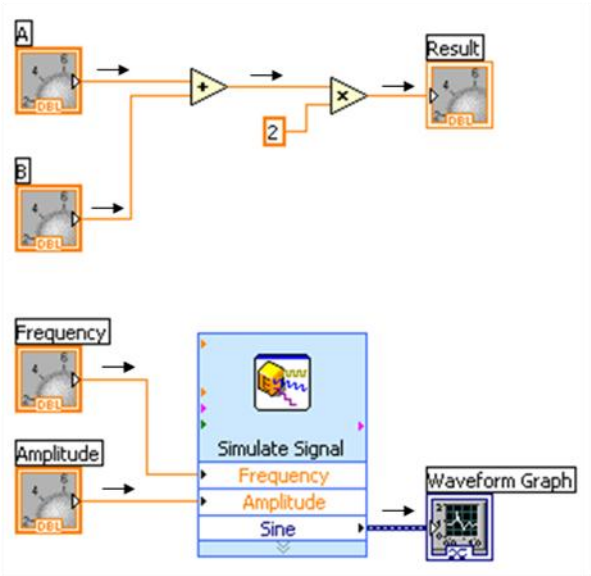
Each terminal contains useful information about the Front Panel object it corresponds to. For example, the color and symbols provide information about the data type. For example: The dynamic data type is a polymorphic data type represented by dark blue terminals. Boolean terminals are green with TF lettering.

In general, blue terminals should wire to blue terminals, green to green, and so on. This is not a hard-and-fast rule; LabVIEW will allow a user to connect a blue terminal (dynamic data) to an orange terminal (fractional value), for example. But in most cases, look for a match in colors. A red dot indicates that values are being converted from one variable type to another.

Controls have an arrow on the right side and have a thick border. Indicators have an arrow on the left and a thin border. Logic rules apply to wiring in LabVIEW: Each wire must have one (but only one) source (or control), and each wire may have multiple destinations (or indicators).

Dataflow Programming

- Block diagram execution
  - Dependent on the flow of data
  - Block diagram does NOT execute left to right
- Node executes when data is available to ALL input terminals
- Nodes supply data to all output terminals when done



LabVIEW follows a dataflow model for running VIs. A block diagram node executes when all its inputs are available. When a node completes execution, it supplies data to its output terminals and passes the output data to the next node in the dataflow path. Visual Basic, C++, JAVA, and most other text-based programming languages follow a control flow model of program execution. In control flow, the sequential order of program elements determines the execution order of a program.

Consider the block diagram above. It adds two numbers and then multiplies by 2 from the result of the addition. In this case, the block diagram executes from left to right, not because the objects are placed in that order, but because one of the inputs of the Multiply function is not valid until the Add function has finished executing and passed the data to the Multiply function. Remember that a node executes only when data are available at all of its input terminals, and it supplies data to its output terminals only when it finishes execution. In the second piece of code, the Simulate Signal Express VI receives input from the controls and passes its result to the Graph.

You may consider the add-multiply and the simulate signal code to co-exist on the same block diagram in parallel. This means that they will both begin executing at the same time and run independent of one another. If the computer running this code had multiple processors, these two pieces of code could run independent of one another (each on its own processor) without any additional coding.

There is no set answer on how to approach writing a program, however there are some useful tips.

Before you start, ask yourself:

What do you actually want you program to do?
What inputs do you have to get you to this event?
How do you combine the inputs to get the desired output?

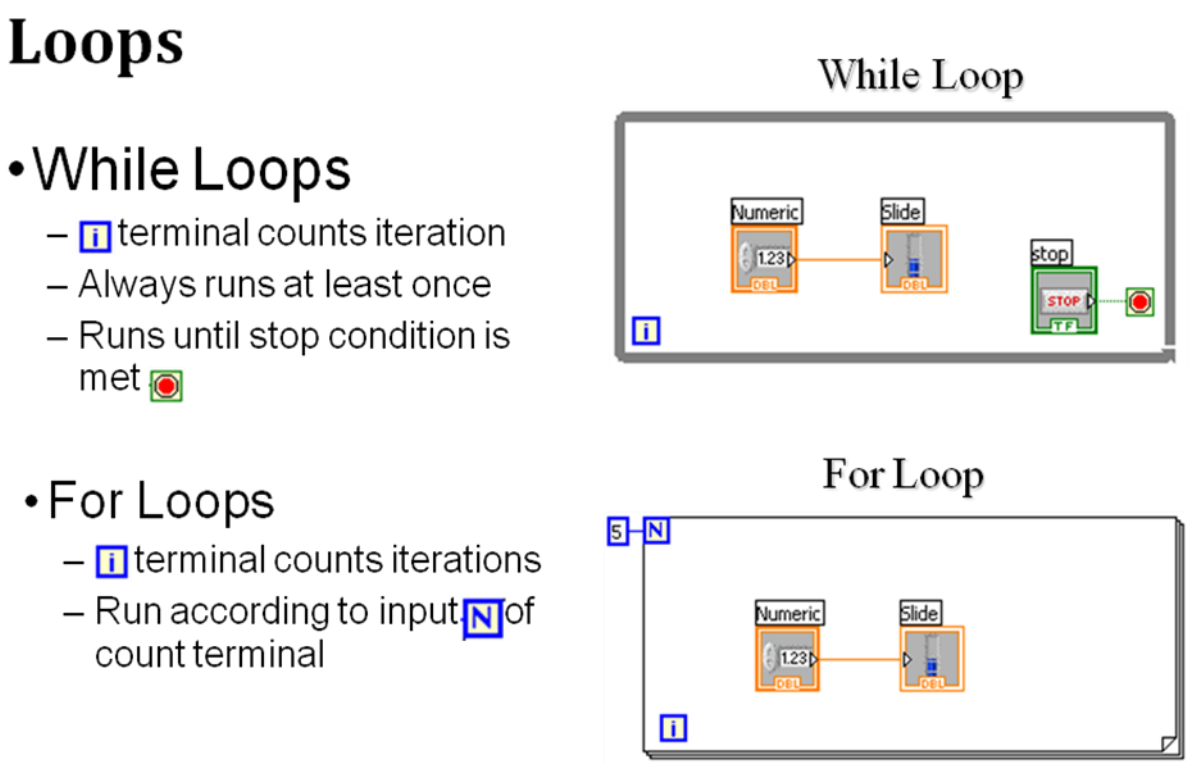
Once you have answered these questions, you can start the process of writing the program. This is generally an iterative process, it is often useful (though not always the case) to start with a simple solution and add addition functions rather than try to do everything at once.

It is also important you document your work as you go, this will help you if you have to come back to it at a later point, or if anyone else has to work with your program.

The general process can be outlined as:

1. Define the problem
2. Plan your solution
3. Code the program
4. Test the program
5. Document the program

Program Elements



6 While and For Loops

Both the While and For Loops are located on the **Functions»Structures** palette. The For Loop differs from the While Loop in that the For Loop executes a set number of times. A While Loop stops executing the subdiagram only if the value at the conditional terminal exists.

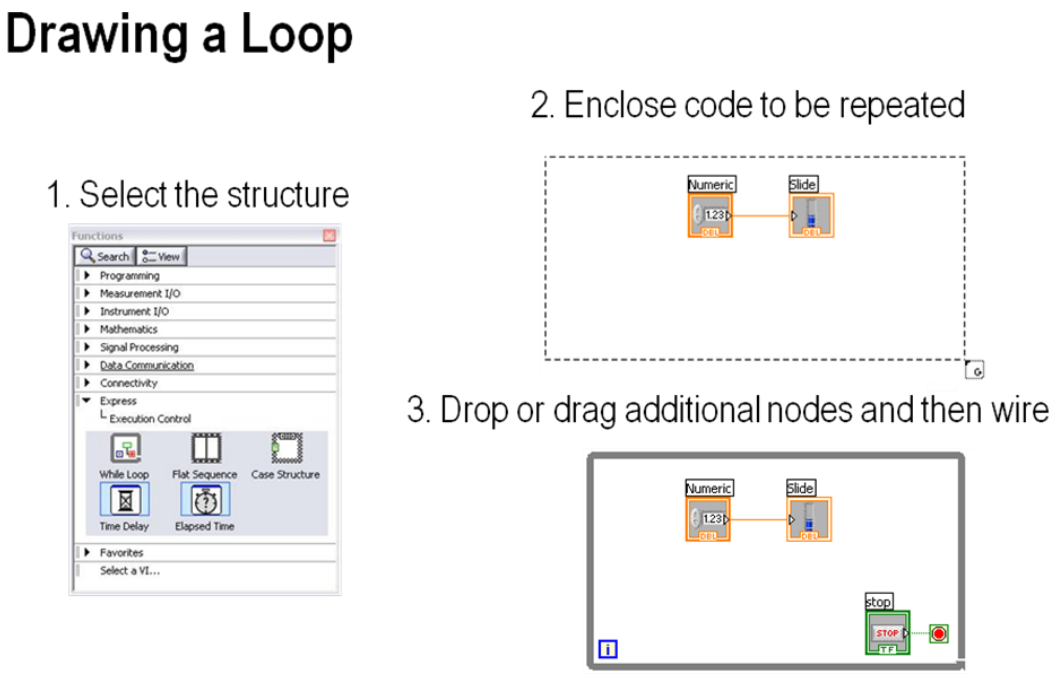
While Loops

Similar to a Do Loop or a Repeat-Until Loop in text-based programming languages, a While Loop, shown at the top right, executes a subdiagram until a condition is met. The While Loop executes the sub diagram until the conditional terminal, an input terminal, receives a specific Boolean value. The default behavior and appearance of the conditional terminal is **Stop If True**. When a conditional terminal is **Stop If True**, the While Loop executes its subdiagram until the conditional terminal receives a TRUE value. The iteration terminal (an output terminal), shown at left, contains the number of completed iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns 0.

For Loops

A For Loop, shown above, executes a subdiagram a set number of times. The value in the count terminal (an input terminal) represented by the N, indicates how many times to repeat the subdiagram. The iteration terminal (an output terminal), shown at left, contains the number of

completed iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns 0.



7 Drawing a Loop

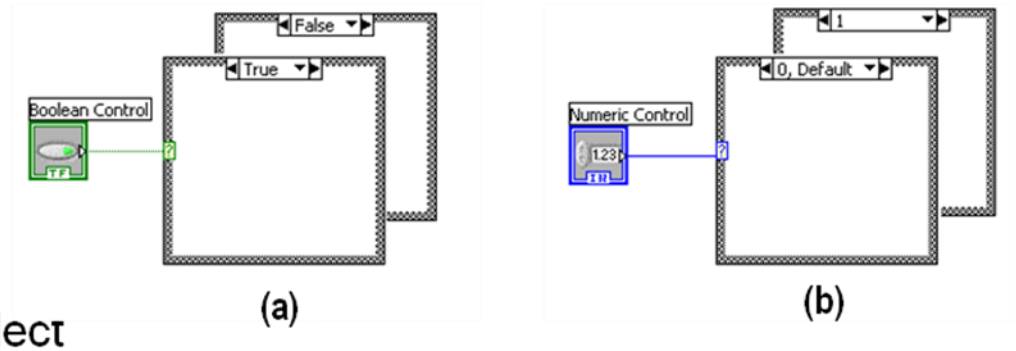
Place loops in your diagram by selecting them from the Structures palette of the Functions palette:

When selected, the mouse cursor becomes a special pointer that you use to enclose the section of code you want to repeat.

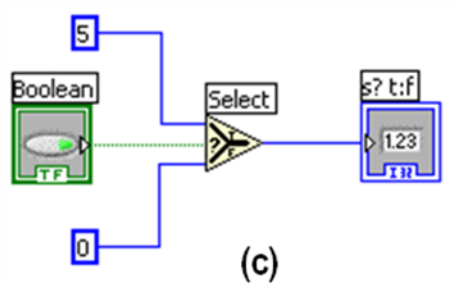
Click the mouse button to define the top-left corner, click the mouse button again at the bottom-right corner, and the While Loop boundary is created around the selected code.

Drag or drop additional nodes in the While Loop if needed.

1. Case Structures



2. Select



8 Different ways of making a choice in LabVIEW

Case Structure

The Case Structure has one or more subdiagrams, or cases, exactly one of which executes when the structure executes. The value wired to the selector terminal determines which case to execute and can be boolean, string, integer, or enumerated type. Right-click the structure border to add or delete cases. Use the Labeling tool to enter value(s) in the case selector label and configure the value(s) handled by each case. It is found at **Functions»Programming»Structures»Case Structure**.

Select

Returns the value wired to the **t** input or **f** input, depending on the value of **s**. If **s** is TRUE, this function returns the value wired to **t**. If **s** is FALSE, this function returns the value wired to **f**. The connector pane displays the default data types for this polymorphic function. It is found at **Functions»Programming» Comparison»Select**.

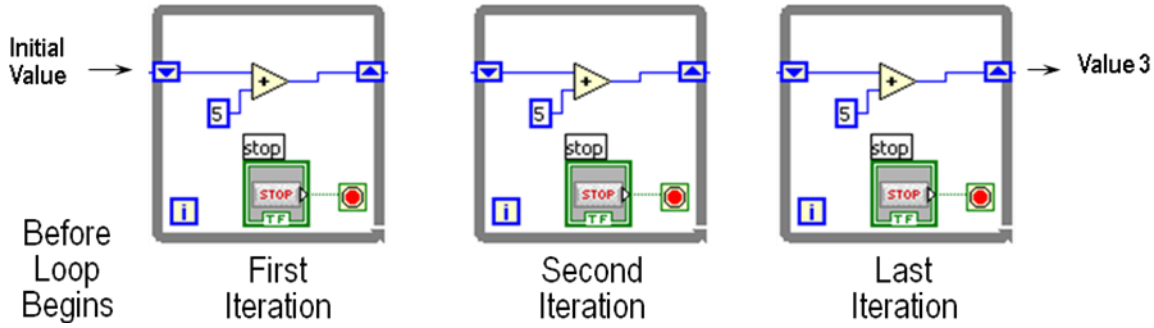
Example a: Boolean input: Simple if-then case. If the Boolean input is TRUE, the true case will execute; otherwise the FALSE case will execute.

Example b: Numeric input. The input value determines which box to execute. If out of range of the cases, LabVIEW will choose the default case.

Example c: When the Boolean passes a TRUE value to the Select VI, the value 5 is passed to the indicator. When the Boolean passes a FALSE value to the Select VI, 0 is passed to the indicator.

Shift Registers

- Available at left or right border of loop structures
- Right-click the border and select **Add Shift Register**
- Right terminal stores data on completion of iteration
- Left terminal provides stored data at beginning of next iteration



Shift registers transfer data from one iteration to the next:

Right-click on the left or right side of a For Loop or a While Loop and select Add Shift Register.

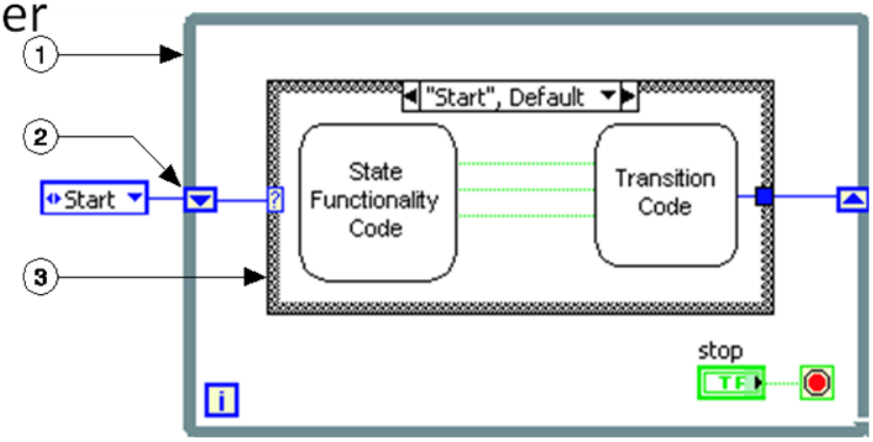
The right terminal stores data at the end of an iteration. Data appears at the left terminal at the start of the next iteration.

A shift register adapts to any data type wired into it.

An input of 0 would result in an output of 5 the first iteration, 10 the second iteration and 15 the third iteration. Said another way, shift registers are used to retain values from one iteration to the next. They are valuable for many applications that have memory or feedback between states.

State Machines

- While Loop
- Case Structure
- Shift Register



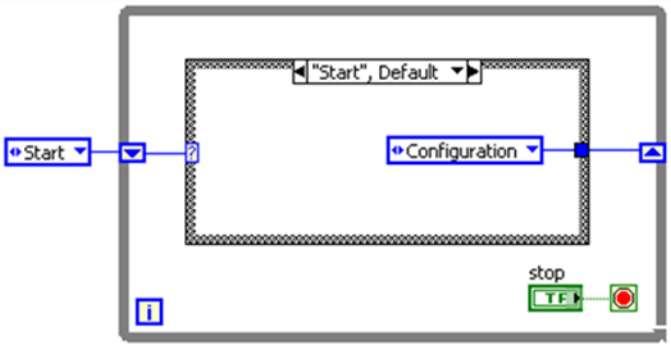
You can use the state machine design pattern to implement an algorithm that you can explicitly described with a state diagram or flowchart. A state machine consists of a set of states and a transition function that maps to the next state.

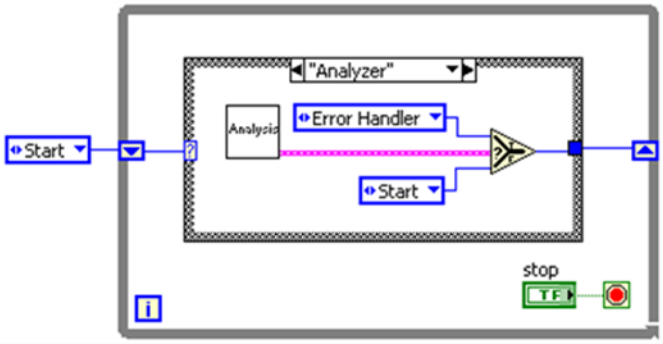
Each state can lead to one or multiple states or end the process flow.

A common application of State machines is to create user interfaces. In a user interface, different user actions send the user interface into different processing segments. Each processing segment acts as a state.

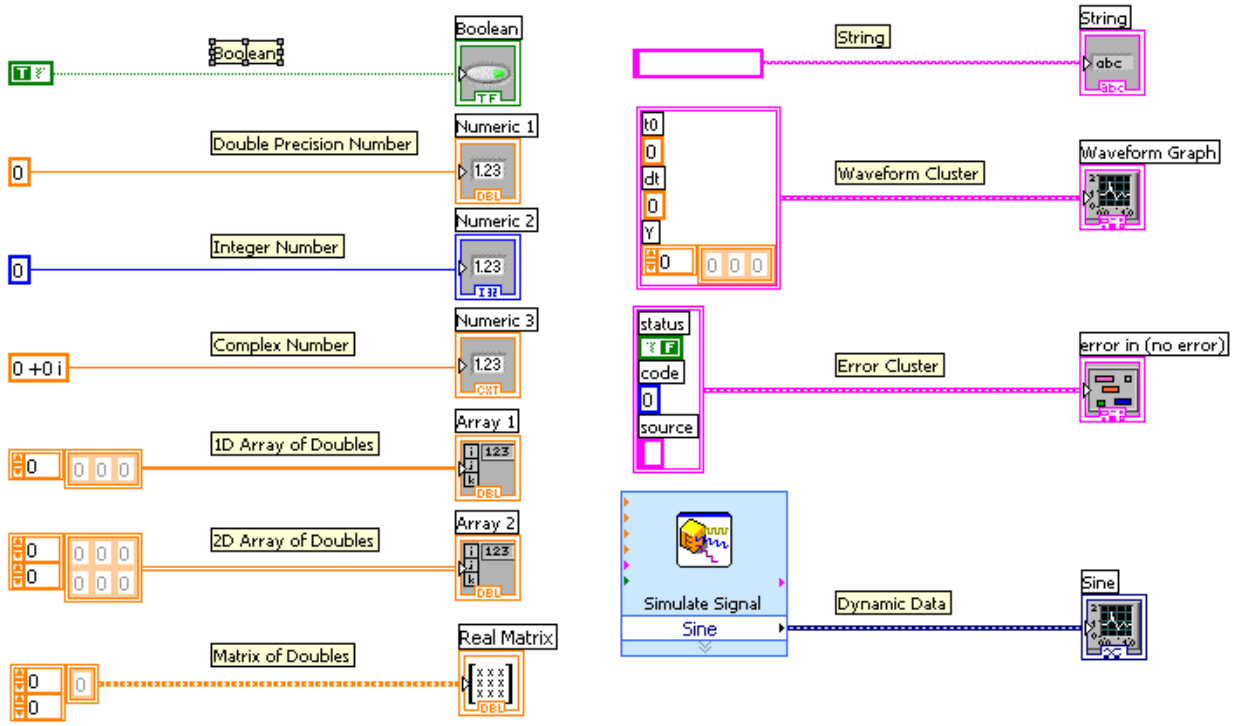
Process testing is another common application of the state machine design pattern. For a process test, a state represents each segment of the process. Depending on the result of each state's test, a different state might be called.

- Several programming techniques exist for transitioning from state to state in LabVIEW using State Machines
- Default transition implies that after one state, another state always follows
- Transitions between two potential states can be handled by a Select Function





Review of datatypes



9 Type of data in LabVIEW

LabVIEW utilizes many common datatypes. These Datatypes include:

Boolean, Numeric, Arrays, Strings, Clusters, and more.

The color and symbol of each terminal indicate the data type of the control or indicator. Control terminals have a thicker border than indicator terminals. Also, arrows appear on front panel terminals to indicate whether the terminal is a control or an indicator. An arrow appears on the right if the terminal is a control, and an arrow appears on the left if the terminal is an indicator.

Definitions

Array: Arrays group data elements of the same type. An array consists of elements and dimensions. Elements are the data that make up the array. A dimension is the length, height, or depth of an array. An array can have one or more dimensions and as many as $(2^{31}) - 1$ elements per dimension, memory permitting.

Cluster: Clusters group data elements of mixed types, such as a bundle of wires in a telephone cable, where each wire in the cable represents a different element of the cluster.

See **Help»Search the LabVIEW Help…** for more information. The *LabVIEW User Manual* on ni.com provides additional reference for data types found in LabVIEW.

Debugging Techniques

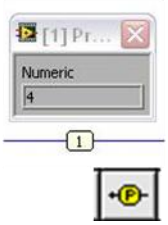
- **Finding Errors**

Click on broken **Run** button.
Window showing error appears.

- **Execution Highlighting**

Click on **Execution Highlighting** button; data flow is animated using bubbles. Values are displayed on wires.

- **Probes**



Right-click on wire to display probe and it shows data as it flows through wire segment.

You can also select Probe tool from Tools palette and click on wire.

When your VI is not executable, a broken arrow is displayed in the Run button in the palette.

Finding Errors: To list errors, click on the broken arrow. To locate the bad object, click on the error message.

Execution Highlighting: Animates the diagram and traces the flow of the data, allowing you to view intermediate values. Click on the **light bulb** on the toolbar.

Probe: Used to view values in arrays and clusters. Click on wires with the **Probe** tool or right-click on the wire to set probes.

Retain Wire Values: Used in conjunction with probes to view the values from the last iteration of the program.

Breakpoint: Set pauses at different locations on the diagram. Click on wires or objects with the **Breakpoint** tool to set breakpoints.

Exercise 1

Complete the following steps to create a VI that acquires data from your sound card.

1. Launch LabVIEW.

2. In the **Getting Started** window, click the **Blank VI** link.

3. Display the block diagram by pressing <Ctrl+E> or selecting **Window»Show Block Diagram**.

4. Place the Acquire Sound Express VI on the block diagram. Right-click to open the functions palette and select **Express»Input»Acquire Sound.** Place the Express VI on the block diagram.

5. In the configuration window under **#Channels**, select **1** from the drop-down list and click "OK".

6. Place the Filter Express VI to the right of the Acquire Signal VI on the block diagram. From the functions palette, select **Express»Signal Analysis**»**Filter** and place it on the block diagram. In the configuration window under **Filtering Type**, choose "**Highpass**." Under Cutoff Frequency, use a value of 300 Hz. Click "OK."

7. Make the following connections on the block diagram by hovering your mouse over the terminal so that it becomes the wiring tool and clicking once on each of the terminals you wish to connect:

a. Connect the "Data" output terminal of the Acquire Signal VI to the "Signal" input of the Filter VI.

b. Create a graph indicator for the filtered signal by right-clicking on the "Filtered Signal" output terminal and choose **Create**»**Graph Indicator.**

8. Return to the front panel by pressing <Ctrl+E> or **Window»Show Front Panel**.

9. Run your program by clicking the run button. Hum or whistle into your microphone and observe the data you acquire from your sound card.

10. Save the VI .

11. Close the VI.

Exercise 2

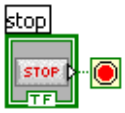
1. Create a VI that produces a sine wave with a specified frequency and displays the data on a Waveform Chart until stopped by the user.

2. Open a blank VI from the Getting Started screen.

3. Place a chart on the front panel. Right-click to open the controls palette and select **Controls»Modern»Graph»Waveform Chart**.

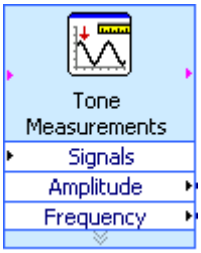
4. Place a dial control on the front panel. From the controls palette, select **Controls»Modern »Numeric»Dial**. Notice that when you first place the control on the front panel, the label text is highlighted. While this text is highlighted, type “Frequency In” to give a name to this control. You will need to edit this control so it gives a range from 10-100 as the Simulate Signal VI only works with frequencies above 10Hz.

5. Go to the block diagram (<Ctrl+E>) and place a while loop down. Right-click to open the functions palette and select **Express»Execution Control»While Loop**. Click and drag on the block diagram to make the while loop the correct size. Select the waveform chart and dial and drag them inside the while loop if they are not already. Notice that a stop button is already connected to the conditional terminal of the while loop.



6. Place the Simulate Signal Express VI on the block diagram. From the functions palette, select **Express»Signal Analysis»Simulate Signal** and place it on the block diagram inside the while loop. In the configuration window under Timing, choose “Simulate acquisition timing.” Click “OK.”

7. Place a Tone Measurements Express VI on the block diagram (**Express»Signal Analysis**»**Tone Measurements**). In the configuration window, choose Amplitude and Frequency measurements in the Single Tone Measurements section. Click “OK.”



8. Make the following connections on the block diagram by hovering your mouse over the terminal so that it becomes the wiring tool and clicking once on each of the terminals you wish to connect:

a. Connect the “Sine” output terminal of the Simulate Signal VI to the “Signals” input of the Tone Measurements VI.

b. Connect the “Sine” output to the Waveform Chart.

c. Create indicators for the amplitude and frequency measurements by right-clicking on each of the terminals of the Tone Measurements Express VI and selecting **Create**»**Numeric Indicator**.

d. Connect the “Frequency In” control to the “Frequency” terminal of the Simulate Signal VI.

9. Return to the front panel and run the VI. Move the “Frequency In” dial and observe the frequency of the signal. Click the stop button once you are finished.

10. Save the VI.

11. Close the VI.

Notes

When you bring up the functions palette, press the small push pin in the upper left hand corner of the palette. This will tack down the palette so that it doesn’t disappear. This step will be omitted in the following exercises, but should be repeated.

Exercise 3

1. Create a VI that measures the frequency and amplitude of the signal from your sound card and displays the acquired signal on a waveform chart. The instructions are the same as in Exercise 2, but the Sound Signal VI is used in place of the Simulate Signal VI. Try to do this without following the instructions!

2. Open a blank VI.

3. Go to the block diagram and place a While Loop down (**Express»Execution Control»While Loop**).

4. Place the Acquire Sound Express VI on the block diagram (**Express»Input» Acquire Sound**).

5. Place a Filter Express VI on the block diagram. In the configuration window choose a highpass filter and a cutoff frequency of 300 Hz.

6. Place a Tone Measurements Express VI on the block diagram (**Express»Signal Analysis**»**Tone**). In the configuration window, choose Amplitude and Frequency measurements in the Single Tone Measurements section.

7. Create indicators for the amplitude and frequency measurements by right-clicking on each of the terminals of the Tone Measurements Express VI and selecting **Create»Numeric Indicator**.

8. Connect the “Data” terminal of the Acquire Sound Express VI to the “Signal” input of the Filter VI.

9. Connect the “Filtered Signal” terminal of the Filter VI to the “Signals” input of the Tone Measurements VI.

10. Create a graph indicator for the Filtered Signal by right-clicking on the “Filtered Signal” terminal and selecting **Create**»**Graph Indicator.**

11. Return to the front panel and run the VI. Observe the signal from your sound card and its amplitude and frequency. Hum or whistle into the microphone and observe the amplitude and frequency you are producing.

12. Save the VI. Close the VI.

Exercise 4 – Decision Making and Saving Data

1. Create a VI that allows you to save your data to file if the frequency of your data goes below a user-controlled limit.

2. Open Exercise 3.

3. Go to **File»Save As…** and save it. In the "Save As" dialog box, make sure **substitute copy for original** is selected and click "Continue…".

4. Add a case structure to the block diagram inside the while loop (**Functions»Programming»Structures»Case Structure**).

5. Inside the "true" case of the case structure, add a Write to Measurement File Express VI (**Functions»Programming»File I/O»Write to Measurement File**).

a. In the configuration window that opens, choose "Save to series of files (multiple files)." Note the default location your file will be saved to and change it if you wish.

b. Click "Settings…" and choose "Use next available file name" under the Existing Files heading.

c. Under **File Termination** choose to start a new file after 10 segments. Click "OK" twice.

6. Add code so that if the frequency computed from the Tone Measurements Express VI goes below a user-controlled limit, the data will be saved to file. **Hint:** Go to **Functions»Programming»Comparison»Less?**

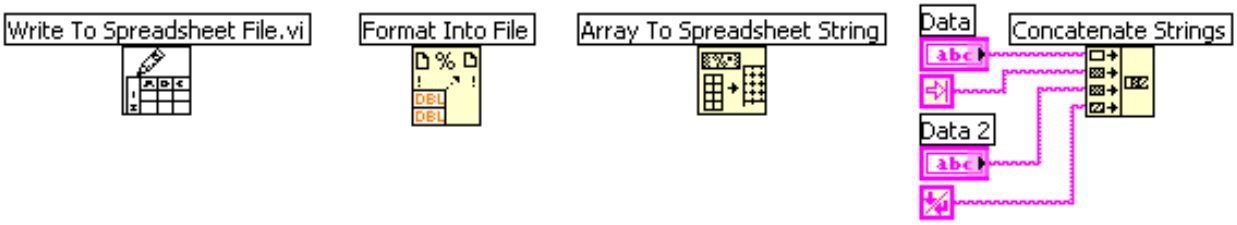
7. Remember to connect your data from the DAQ Assistant or Acquire Sound Express VI to the "Signals" input of the Write to Measurement File VI. If you need help, refer to the solution to this exercise.

8. Go to the front panel and run your VI. Vary your frequency limit and then stop the VI.

9. Navigate to **My Documents»LabVIEW Data** and open one of the files that was saved there. Examine the file structure and check to verify that 10 segments are in the file.

10. Save your VI and close it.

Spreadsheet Vis



Sometimes it is better to save data as text so you can import it into Excel. Spreadsheets are usually ASCII (text) files with a certain type of formatting. Two formatting methods are comma separated values (CSV) and tab delimited. Tab delimited files, which are the most popular, have tabs constants between columns of data and end of line constants between rows. LabVIEW includes VIs that perform this formatting:

- **Write to Spreadsheet File** takes either 1D or 2D arrays of numeric data, formats this data, and writes this information to file.
- **Format Into File** takes many different types of data (string, numeric, Boolean) and writes this information to file, using either a file path or file reference. This function can be resized to include as many data terminals as necessary.
- **Array to Spreadsheet String** is a string function that formats array data into a string that can be written to a text file.
- The **Concatenate String** function is used to create longer strings from shorter ones and is the most flexible when converting data to a string that can be written to a text file.

Exercise 5– Spreadsheet Vis

(not all wiring instructions are given here – some are obvious – refer to the diagram or ask a demonstrator if you are unsure)

1. Open a blank new VI from the Getting Started screen.

2. Place the Open/Create/Replace File function on the block diagram. Right-click on the block diagram to open the functions palette and select File I/O » Open/Create/Replace File.

3. Right-click the **operation** terminal of the Open/Create/Replace File function and select **Create » Constant** from the shortcut menu, and select **open or create** from the drop down menu.

4. Place a While loop from the Structures palette on the block diagram to the right of the Open/Create/Replace File function. Right-click on the block diagram select Structures » While Loop.

5. Place a Write Text File function inside the While Loop. Right-click on the block diagram select File I/O » Write To Text File.

6. Wire the **refnum out** terminal from the Open/Create/Replace File function to the **file (use dialog)** terminal of the Write Text File function.

7. Wire the **error out** terminal from the Open/Create/Replace File function to the **error in** terminal of the Write Text File function.

8. Place an Array to Spreadsheet String function inside the while loop and to the left of the on Open/Create/Replace File function. Right-click on the block diagram and select String » Array to Spreadsheet String.

9. Right-click the **format string** terminal of the Array to Spreadsheet function and select **Create » Constant** from the shortcut menu and enter **"%0.4f"** in the string constant to format the input data.

10. Place a Build Array Function on the block diagram. Right-click on the block diagram and select Array » Build Array.

11. Place a Random Number inside the While Loop. Right-click on the block diagram and select Numeric » Random Number (0-1).

12. Wire the **error out** terminal of the Write Text File function to an output tunnel on the While Loop.

13. Place an Unbundle By Name function inside the While Loop. Right-click on the block diagram to open the functions palette and select **Cluster & Variant » Unbundle By Name.**

14. Wire the error out from the Write Text File function to the Unbundle By Name function.

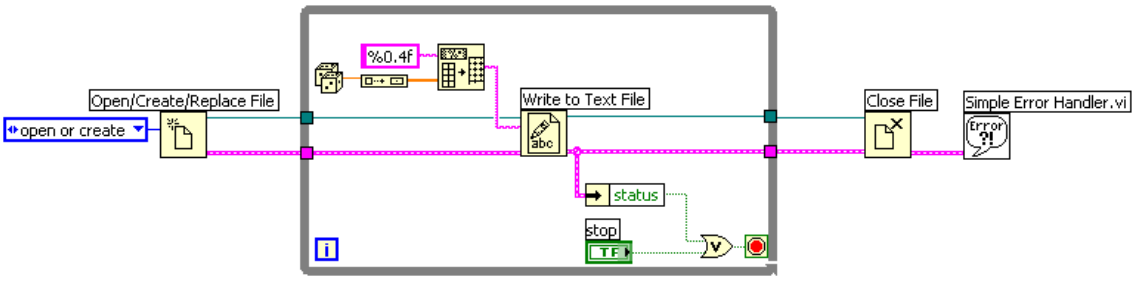
15. Place an Or function in the While Loop. Right-click on the block diagram to open the functions palette and select **Boolean » Or**.

16. Switch to the front panel and place a stop button. Right-Click on the front panel to open the Controls palette and select **Boolean » Stop Button**.

17. On the block diagram, wire the **status** element of the error cluster to the **x** input of the Or function and wire the stop button to the **y** input.

18. Wire the output of the Or function to the conditional terminal of the While Loop.

19. Place a Close File function to the right of the While Loop. Right-click on the block diagram to open the functions palette and select **File I/O » Close File**.



20. Wire the refnum output tunnel to the refnum input terminal of the Close File function.

21. Wire the error output tunnel to the error in terminal of the Close File function.

22. Return to the front panel and run the VI. You will be prompted to “Choose or enter path of file to open”, enter: “spreadsheet.xls”.

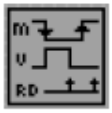
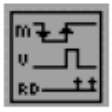
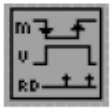
23. Click on the stop button to stop the execution of the VI.

24. Open the file named: “spreadsheet.xls”.

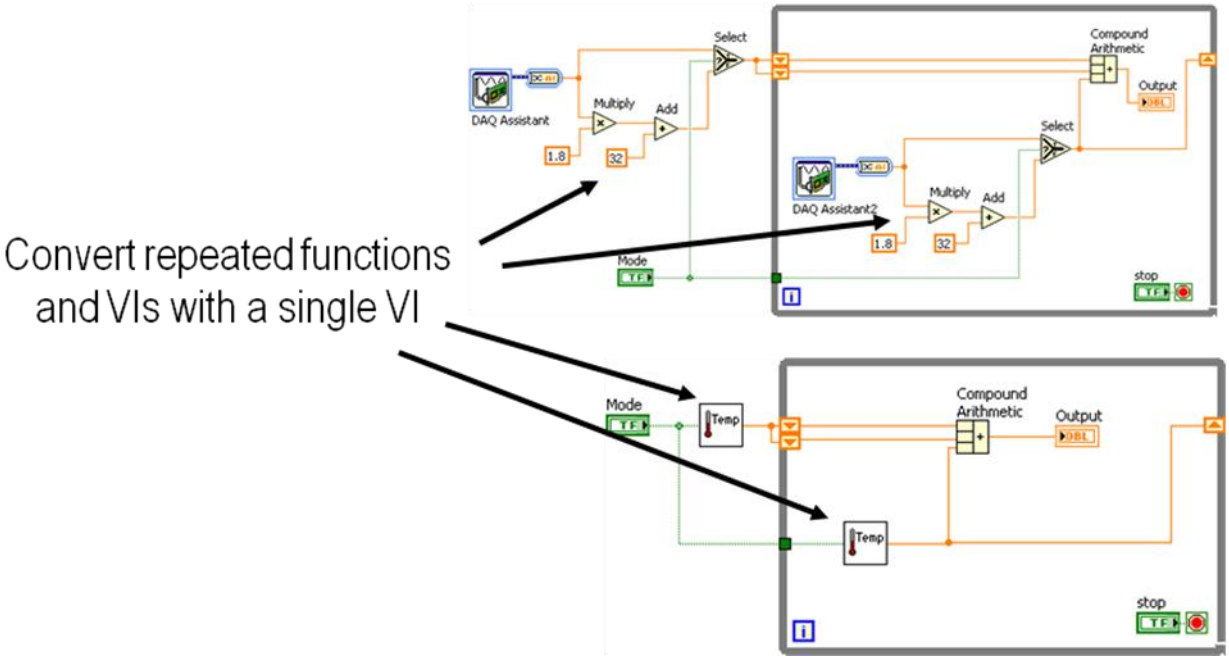
25. Save and the close the VI.

Switches

One of the most basic control mechanisms is through switches. However, depending on how you want the switch to behave there are various “mechanical options” to choose from:

Icon	Latch?	Action on Press or Release?	How many times will LabVIEW read it?
	YES	PRESS	Many times
	YES	RELEASE	Many times
	NO	PRESS/RELEASE	Maybe never (released too quick) Maybe many times (released “slowly”)
	NO	PRESS	Once - revert when LabVIEW reads
	NO	RELEASE	Once - revert when LabVIEW reads
	NO	PRESS	At least once, maybe more - revert when LabVIEW reads after button released.

Sub VIs



Modularity defines the degree to which your VI is composed of discrete components such that a change to one component has minimal impact on other components. In LabVIEW these separate components are called subVIs. Creating subVIs out of your code increases the readability and reusability of your VIs.

In the upper image, we see repeated code allowing the user to choose between temperature scales. Since this portion of this code is identical in both cases, we can create a subVI for it. This will make the code more readable, by being less clustered, and will allow us to reuse code easily. As you can see, the code is far less cluttered now, achieves the exact same functionality and if needed, the temperature scale selection portion of the code can be reused in other applications very easily.

Any portion of LabVIEW code can be turned into a subVI that in turn can be used by other LabVIEW code.

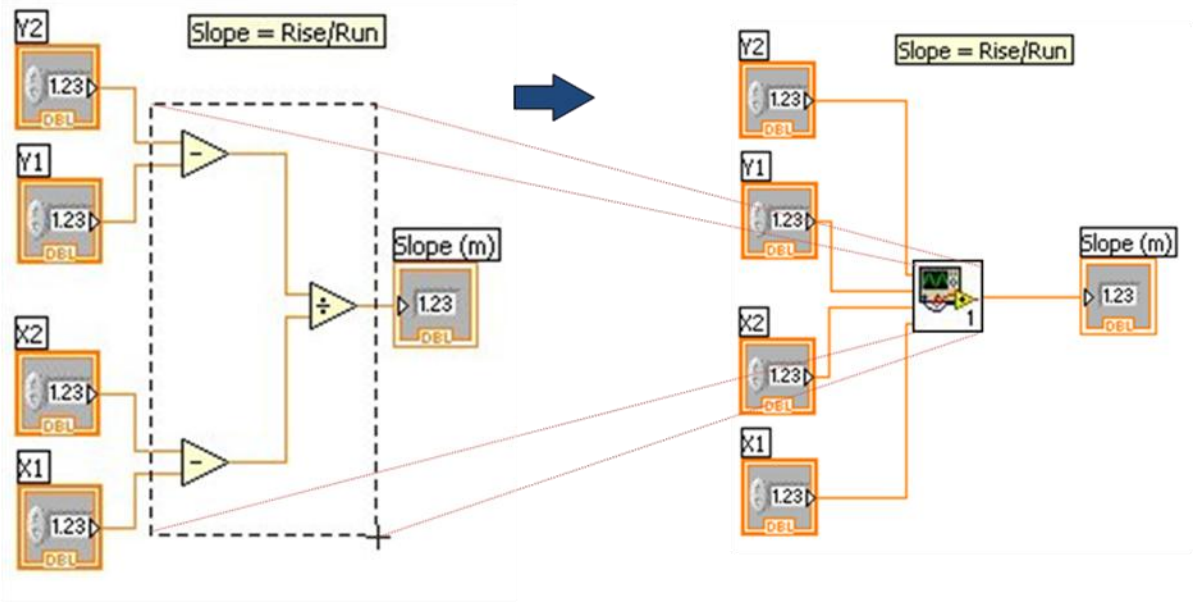
Creating SubVIs

A subVI node corresponds to a subroutine call in text-based programming languages. A block diagram that contains several identical subVI nodes calls the same subVI several times.

The subVI controls and indicators receive data from and return data to the block diagram of the calling VI. Click the **Select a VI** icon or text on the Functions palette, navigate to and double-click a VI, and place the VI on a block diagram to create a subVI call to that VI.

A subVI input and output terminals and the icon can be easily customized. Follow the instructions below to quickly create a subVI.

- Enclose area to be converted into a subVI.
- Select **Edit»Create SubVI** from the Edit Menu.



Creating SubVIs from Sections of a VI

Convert a section of a VI into a subVI by using the Positioning tool to select the section of the block diagram you want to reuse and selecting **Edit»Create SubVI**. An icon for the new subVI replaces the selected section of the block diagram. LabVIEW creates controls and indicators for the new subVI, automatically configures the connector pane based on the number of control and indicator terminals you selected, and wires the subVI to the existing wires.

See Help»Search the LabVIEW Help…»SubVIs for more information.

LabVIEW Functions and SubVIs operate like Functions in other languages

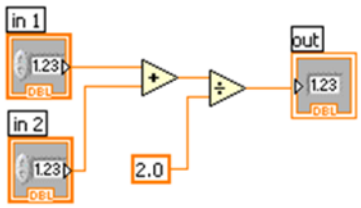
Function Pseudo Code

```
function average (in1, in2, out)
{
out = (in1 + in2)/2.0;
}
```

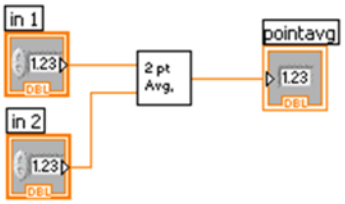
Calling Program Pseudo Code

```
main
{
average (in1, in2, pointavg)
}
```

SubVI Block Diagram



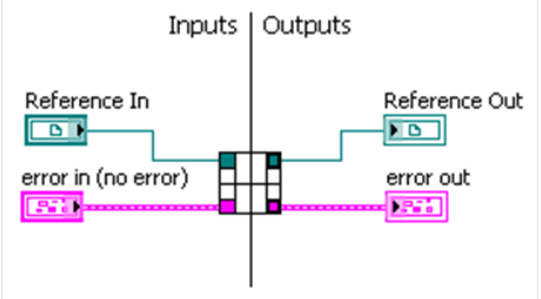
Calling VI Block Diagram



A subVI node corresponds to a subroutine call in text-based programming languages. The node is not the subVI itself, just as a subroutine call statement in a program is not the subroutine itself. A block diagram that contains several identical subVI nodes calls the same subVI several times. The modular approach makes applications easier to debug and maintain. The functionality of the subVI does not matter for this example. The important point is the passing of two numeric inputs and one numeric output.

Icon and Connector Pane

- Use this connector pane layout as a standard
- Top terminals are usually reserved for references, such as a file reference
- Bottom terminals are usually reserved for error clusters



The Icon and Connector Pane allows you to define the data being transferred in and out of the subVI as well as its appearance in the main LabVIEW code. Every VI displays an icon in the upper-right corner of the front panel and block diagram windows. After you build a VI, build the icon and the connector pane so you can use the VI as a subVI.

The icon and connector pane correspond to the function prototype in text-based programming languages. There are many options for the connector pane, but some general standards are specified above. Namely, to always reserve the top terminals for references and the bottom terminals for error clusters.

To define a connector pane, right-click the icon in the upper right corner of the front panel and select **Show Connector** from the shortcut menu. Each rectangle on the connector pane represents a terminal. Use the terminals to assign inputs and outputs. Select a different pattern by right-clicking the connector pane and selecting **Patterns** from the shortcut menu.

An icon is a graphical representation of a VI. If you use a VI as a subVI, the icon identifies the subVI on the block diagram of the VI. The Icon Editor is a utility that comes built into LabVIEW 8 to allow users to fully customize the appearance of their subVIs. This allows programmers to visually distinguish their subVIs, which will greatly improve the usability of the subVI in large portions of code.

After you've defined the connector pane and have customized the icon, you are ready to place the subVI into other LabVIEW code. There are two ways to accomplish this:

To place a subVI on the block diagram

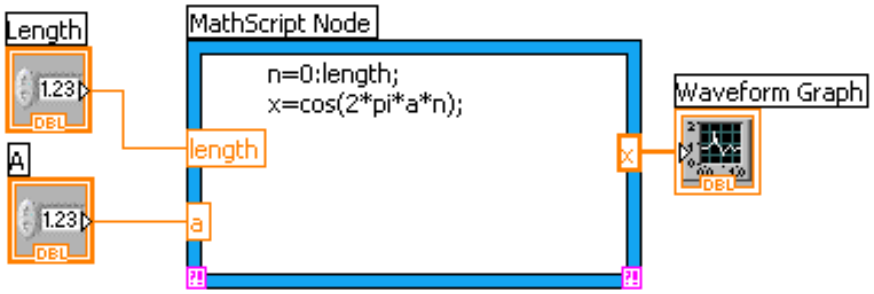
1. Click the **Select a VI** button on the **Functions** palette
2. Navigate to the VI you want to use as a subVI
3. Double-click to place it on the block diagram

To place an open VI on the block diagram of another open VI

1. Use the Positioning tool to click the icon of the VI you want to use as a subVI
2. Drag the icon to the block diagram of the other VI

Math nodes

You can also enter mathematical equations in LabVIEW in so-called MathScript windows. You can place these from the structures window. Finish each line with a semicolon top ensure that the windows only terminates when every line has processed.



10 A MathScript Node

Help for the environment can be accessed using the Mathscript Interactive Environment Window. Type **Help** in the command window for an introduction to MathScript help. **Help** followed by a **function** will display help specific to that function.

File I/O

File I/O operations pass data from memory to and from files. In LabVIEW, you can use File I/O functions to:

- Open and close data files
- Read data from and write data to files
- Read from and write to spreadsheet-formatted files
- Move and rename files and directories
- Change file characteristics
- Create, modify, and read a configuration file

The different file formats that LabVIEW can use or create are the following:

Binary – Binary files are the underlying file format of all other file formats.

ASCII – An ASCII file is a specific type of binary file that is a standard used by most programs. ASCII file are also called text files.

LVM – The LabVIEW measurement data file (.lvm) is a tab-delimited text file you can open with a spreadsheet application or a text-editing application. This file format is a specific type of ASCII file created for LabVIEW. The .lvm file contain information about the data, such as the date and time the data was generated.

TDM – This file format is a specific type of binary created for National Instruments products. It actually consists of two separate files: an XML section contains the data attributes, and a binary file for the waveform.

High Level File I/O: These functions provide a higher level of abstraction to the user by opening and closing the file automatically before and after reading or writing data. Some of these functions are:

- **Write to Spreadsheet File** – Converts a 1D or 2D array of single-precision numbers to a text string and writes the string to a new ASCII file or appends the string to an existing file.
- **Read From Spreadsheet File** – Reads a specified number of lines or rows from a numeric text file beginning at a specified character offset and converts the data to a 2D single-precision array of numbers. The VI opens the file before reading from it and closes it afterwards.
- **Write to Measurement File** – Express VI that writes data to a text-based measurement file (.lvm) or a binary measurement file (.tdm) format.
- **Read from Measurement File** – An Express VI that writes data to a text-based measurement file (.lvm) or a binary measurement file (.tdm) format. You can specify the file name, file format and segment size.

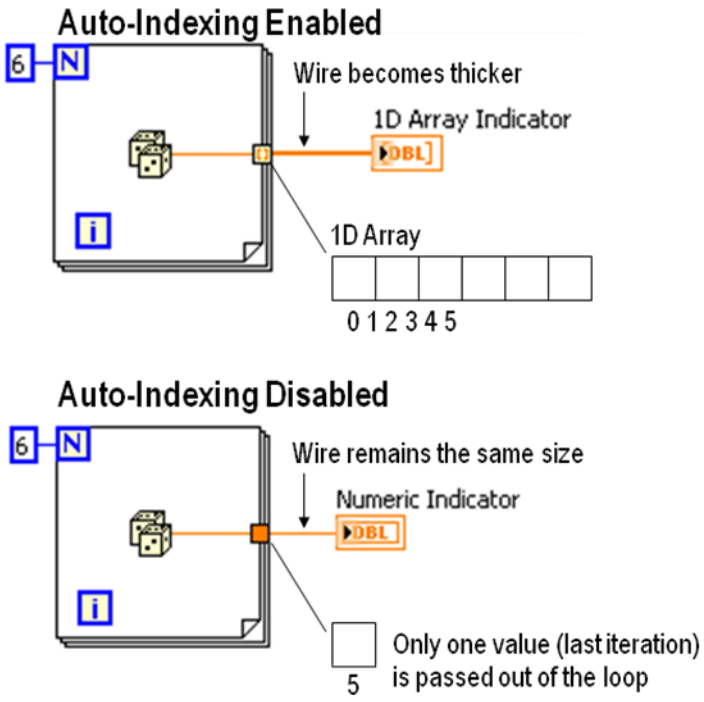
These functions are very easy to use and are excellent for simple applications. In the case where you will do constant streaming to the files by continuously writing to or reading from the file, there may be some overhead in using these functions.

In the next example we will examine how to write to or read from LabVIEW Measurements files (*.lvm files).

Arrays

Building arrays with Loops

- Loops can accumulate arrays at their boundaries with auto-indexing
- For Loops auto-index by default
- While Loops output only the final value by default
- Right-click tunnel and enable/disable auto-indexing



11 How to auto-index arrays

For Loops and While Loops can index and accumulate arrays at their boundaries. This is known as auto-indexing.

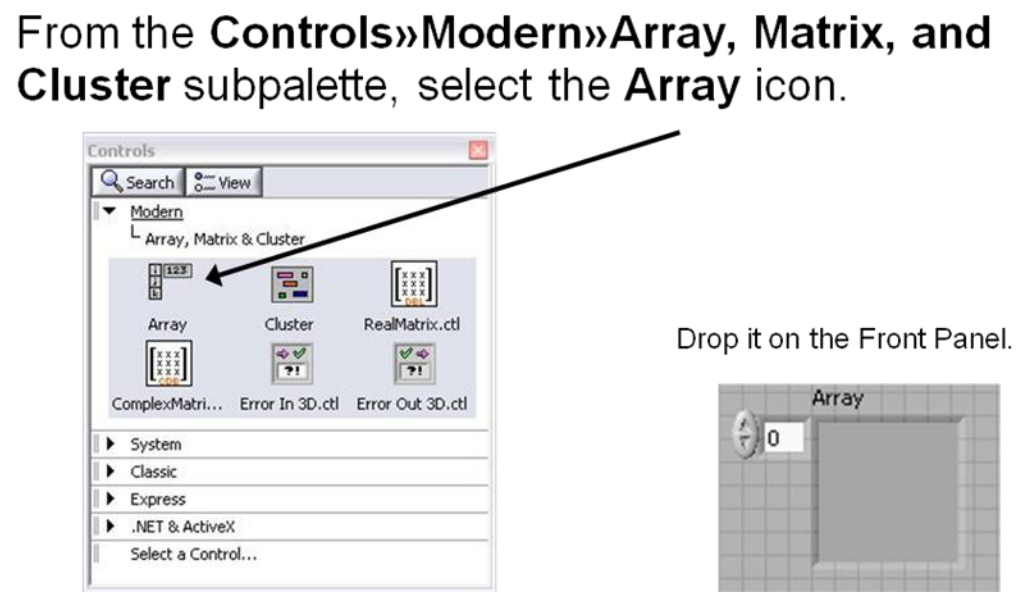
- The indexing point on the boundary is called a tunnel.
- The For Loop default is auto-indexing enabled.
- The While Loop default is auto-indexing disabled.

Examples:

Enable auto-indexing to collect values within the loop and build the array. All values are placed in array upon exiting loop.

Disable auto-indexing if you are interested only in the final value.

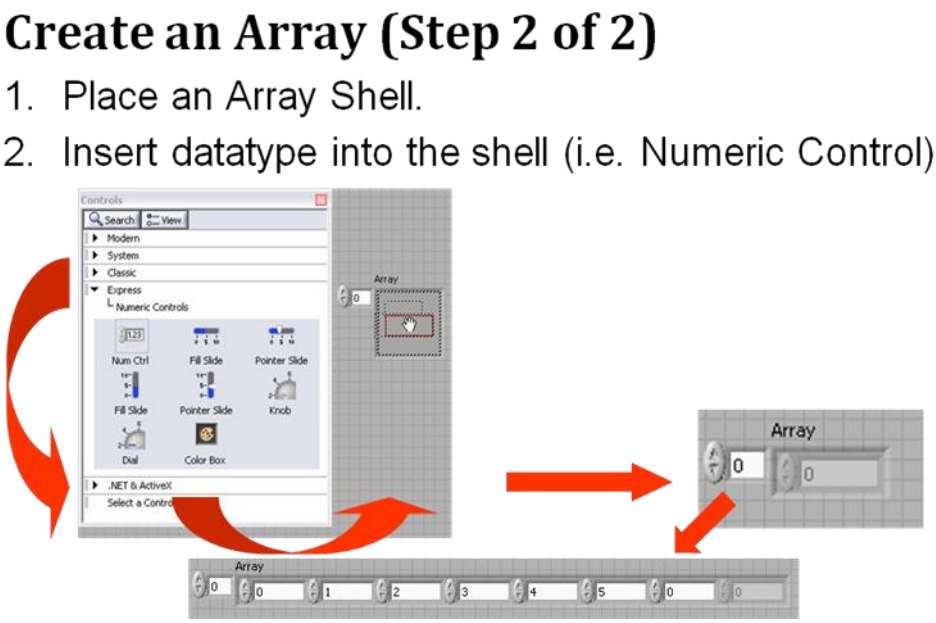
Creating an Array Control



12 An Array Control

To create an array control or indicator as shown, select an array on the **Controls»Modern»Array, Matrix, and Cluster** palette, place it on the front panel, and drag a control or indicator into the array shell. If you attempt to drag an invalid control or indicator such as an XY graph into the array shell, you are unable to drop the control or indicator in the array shell.

You must insert an object in the array shell before you use the array on the block diagram. Otherwise, the array terminal appears black with an empty bracket.

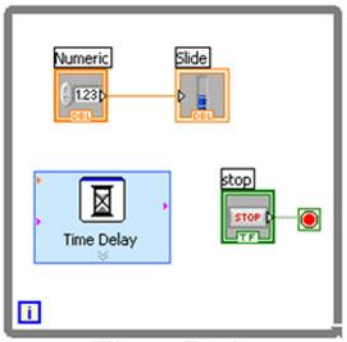


13 Creating an Array

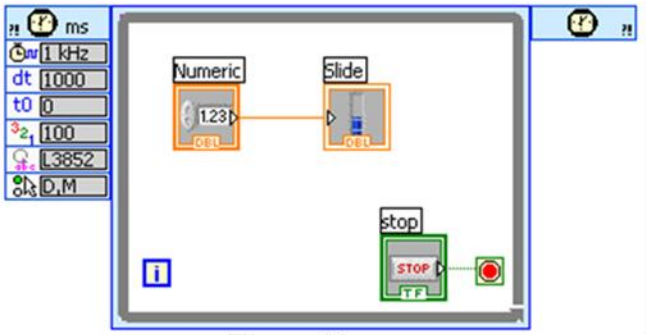
To add dimensions to an array one at a time, right-click the index display and select **Add Dimension** from the shortcut menu. You also can use the Positioning tool to resize the index display until you have as many dimensions as you want.

Timing a loop

1. Loop Time Delay
   - Configure the Time Delay Express VI for seconds to wait each iteration of the loop (works on For and While loops).
2. Timed Loops
   - Configure special timed While loop for desired *dt*.



Time Delay



Timed Loop

Time Delay

The Time Delay Express VI delays execution by a specified number of seconds. Following the rules of Data Flow Programming, the while loop will not iterate until all tasks inside of it are complete, thus delaying each iteration of the loop.

Timed Loops

Executes each iteration of the loop at the period you specify. Use the Timed Loop when you want to develop VIs with multi-rate timing capabilities, precise timing, feedback on loop execution, timing characteristics that change dynamically, or several levels of execution priority.

Double-click the Input Node or right-click the Input Node and select **Configure Timed Loop** from the shortcut menu to display the Loop Configuration dialog box, where you can configure the Timed Loop. The values you enter in the **Loop Configuration** dialog box appear as options in the Input Node.

Wait Until Next ms Multiple

Waits until the value of the millisecond timer becomes a multiple of the specified **millisecond multiple**. Use this function to synchronize activities. You can call this function in a loop to control the loop execution rate. However, it is possible that the first loop period might be short. This function

makes asynchronous system calls, but the nodes themselves function synchronously. Therefore, it does not complete execution until the specified time has elapsed.

Functions»Programming»Timing»Wait Until Next ms Multiple

14 The Wait until Next ms Multiple function

Clusters

Clusters group like or unlike components together. They are equivalent to a *record* in Pascal or a *struct* in C.

Cluster components may be of different data types.

Examples:

Error information—Grouping a Boolean error flag, a numeric error code, and an error source string to specify the exact error.

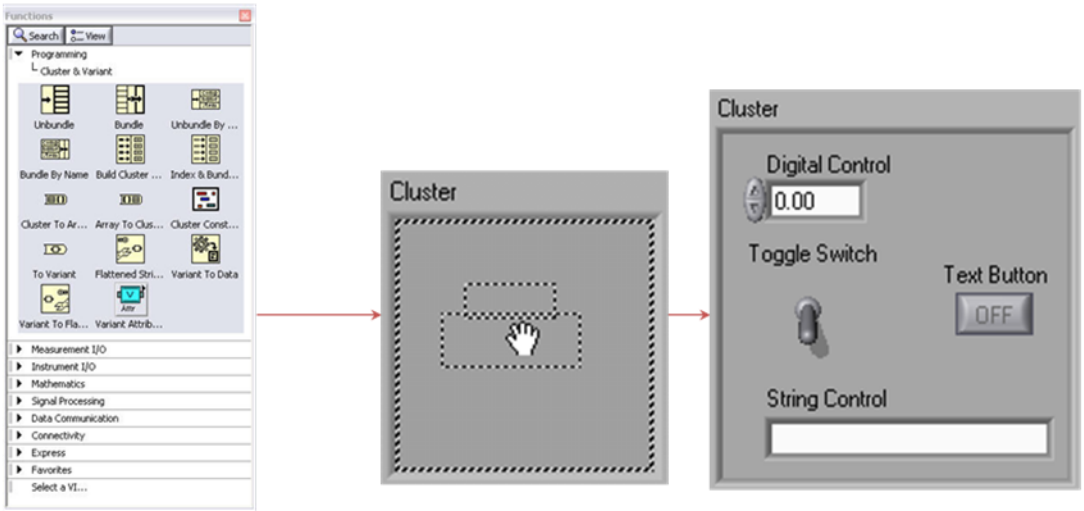
User information—Grouping a string indicating a user's name and an ID number specifying their security code.

All elements of a cluster must be either controls or indicators. You cannot have a string control and a Boolean indicator. Clusters can be thought of as grouping individual wires (data objects) together into a cable (cluster).

Creating a Cluster

1. Select a **Cluster** shell. 2. Place objects inside the shell.

Controls»Modern»Array, Matrix & Cluster



Cluster front panel objects can be created by choosing **Cluster** from the **Controls»Modern»Array, Matrix & Cluster** palette.

This option gives you a shell (similar to the array shell when creating arrays).

You can size the cluster shell when you drop it.

Right-click inside the shell and add objects of any type.

Note: You can even have a cluster inside of a cluster.

The cluster becomes a control or an indicator cluster based on the first object you place inside the cluster.

You can also create a cluster constant on the block diagram by choosing **Cluster Constant** from the **Cluster** palette.

This gives you an empty cluster shell.

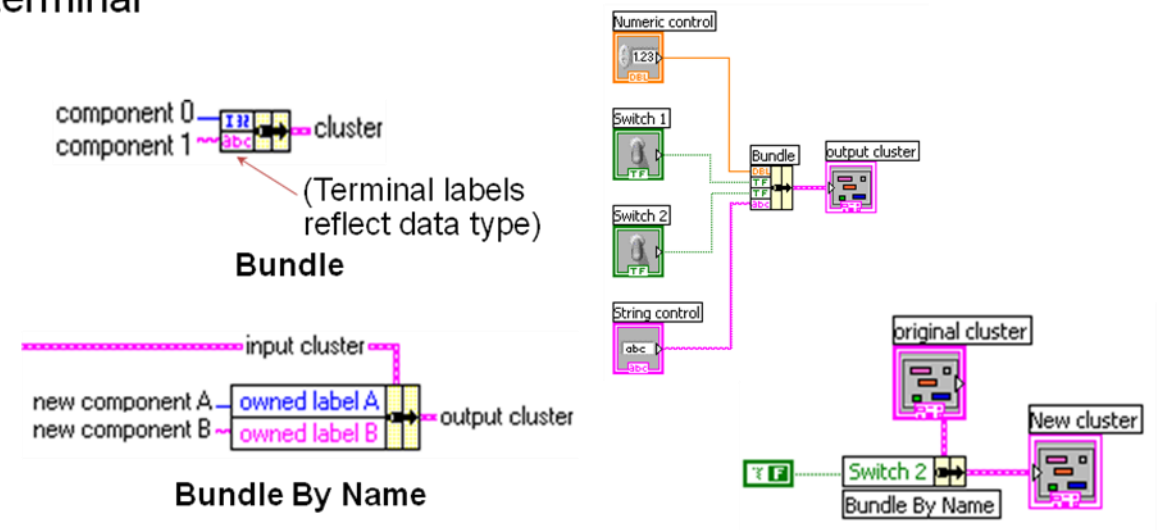
You can size the cluster when you drop it.

Put other constants inside the shell.

Note: You cannot place terminals for front panel objects in a cluster constant on the block diagram, nor can you place “special” constants like the Tab or Empty String constant within a block diagram cluster shell.

Cluster Functions

- In the **Cluster & Variant** subpalette of the **Programming** palette
- Can also be accessed by right-clicking the cluster terminal



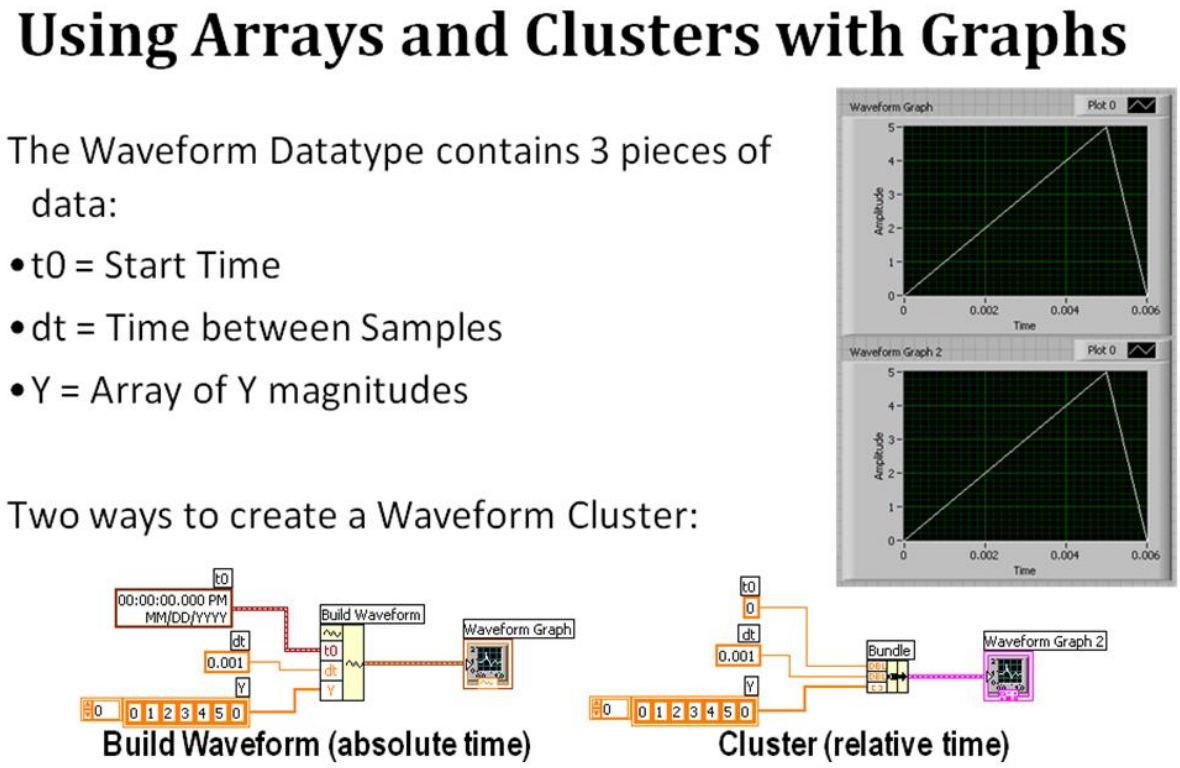
The terms Bundle and Cluster are closely related in LabVIEW.

Example: You use a Bundle Function to create a Cluster. You use an Unbundle function to extract the parts of a cluster.

Bundle function—Forms a cluster containing the given objects (explain the example).

Bundle by Name function—Updates specific cluster object values (the object must have an owned label).

Note: You must have an existing cluster wired into the middle terminal of the function to use Bundle By Name.

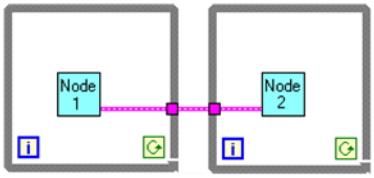


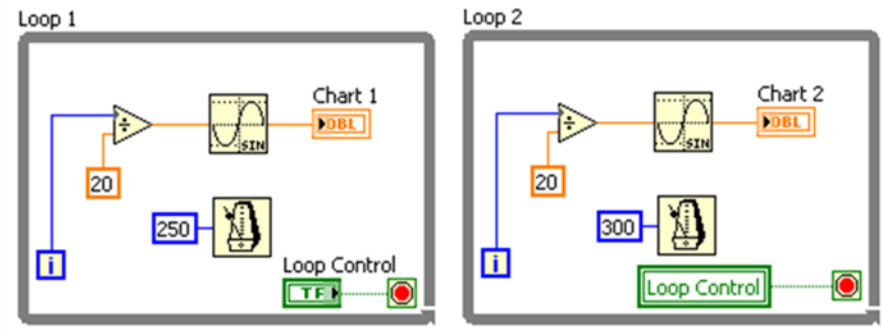
The waveform data type carries the data, start time, and Δ*t* of a waveform. You can create waveforms using the Build Waveform function. Many of the VIs and functions you use to acquire or analyze waveforms accept and return the waveform data type by default. When you wire a waveform data type to a waveform graph or chart, the graph or chart automatically plots a waveform based on the data, start time, and Δ*x* of the waveform. When you wire an array of waveform data types to a waveform graph or chart, the graph or chart automatically plots all the waveforms.

The feedback node is another representation of the same concept. (pictured below) Both programs pictured behave the same.

Communicating between loops

- Communicating between loops using data flow is not possible
  - The left loop will execute completely before the right loop
- Variables are needed when communication with wires does not give the desired behavior

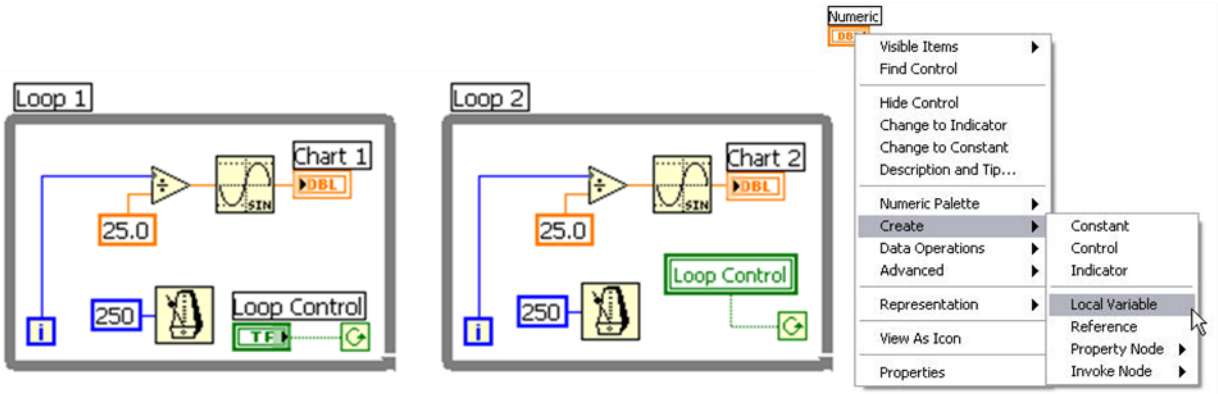




There is no way to communicate between parallel loops using data flow. Data cannot enter or leave a structure while it's still running via dataflow. *Variables* are block diagram elements that allow you to access or store data in another location. *Local variables* store data in front panel controls and indicators. Variables allow you to circumvent normal dataflow by passing data from one place to another without connecting the two places with a wire

Local Variables

- Local Variables allow data to be passed between parallel loops.
- A single control or indicator can be read or written to from more than one location in the program
  - Local Variables break the dataflow paradigm and should be used sparingly



Local variables are located in the **Structures** subpalette of the **Functions** palette.

When you place a local variable on the diagram, it contains by default the name (owned label) of the first object you placed on the front panel.

You use a local variable by first selecting the object you want to access. You can either click on the local variable with the Operating tool and select the object (by owned label) you want to access, or pop up on the local variable and choose the object from the **Select Item** menu.

Next, you must decide to either read or write to the object. Right click on the local variable and choose **Change To Read** or **Change to Write**.

Further Help

LabVIEW has a very comprehensive help system. You should use it! Remember that you can also right click on any vi to see its wiring diagam

Exercise 6 – Manual Analysis

1. Create a VI that displays simulated data on a waveform graph and measures the frequency and amplitude of that data. Use cursors on the graph to verify the frequency and amplitude measurements.

2. Open Exercise 3.

3. Save the VI as “Exercise 6.vi”.

4. Go to the block diagram and remove the While Loop. Right-click the edge of the loop and choose Remove While Loop so that the code inside the loop does not get deleted.

5. Delete the stop button.

6. Make the cursor legend viewable on the graph. Right-click on the graph and select Visible Items»Cursor Legend.

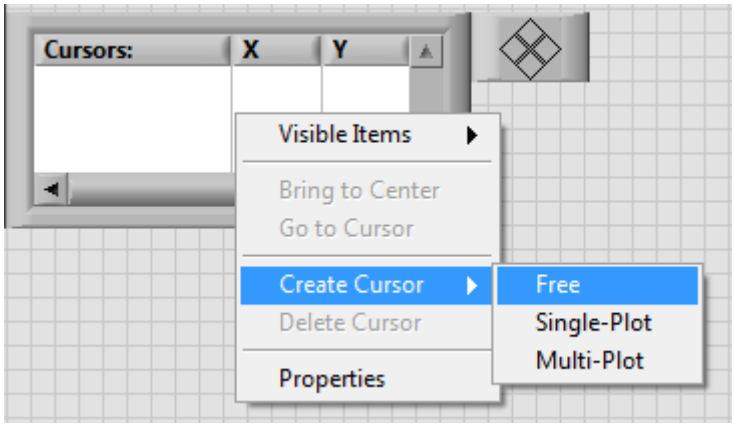
7. Change the maximum value of the “Frequency In” dial to 100. Double-click on the maximum value and type “100” once the text is highlighted.

8. Set a default value for the “Frequency In” dial by setting the dial to the value you would like, right-clicking the dial, and selecting Data Operations»Make Current Value Default.

9. Run the VI and observe the signal on the waveform graph. If you cannot see the signal, you may need to turn on auto-scaling for the x-axis. Right-click on the graph and select X Scale»AutoScale X.

10. Change the frequency of the signal so you can see a few periods on the graph.

11. Manually measure the frequency and amplitude of the signal on the graph using cursors. To make the cursors display on the graph, click on one of the three buttons in the cursor legend. Once the cursors are displayed, you can drag them around on the graph and their coordinates will be displayed in the cursor legend. If you can’t see the cursor check they aren’t at the far edge of the plot window.



12. Remember that the frequency of a signal is the reciprocal of its period (f = 1/T). Does your measurement match the frequency and amplitude indicators from the Tone Measurements VI

13. Save your VI and close it.

14. *If you are making good progress try to see if you can programmatically calculate the frequency from the cursors. (Hint: - you will need to create a property node on the graph indicator for the cursors and select each of two cursors in turn).

Exercise 7 – MathScript

Note that in Mathscript the input to the formula node cannot be a global variable. Make sure that you wire the input to "In1" and then define a global variable and place the value of "In1" in it.

1. Create a VI that uses the MathScript Node to alter your simulated signal and graph it. Use the Interactive MathScript Window to view and alter the data and then load the script you have created back into the MathScript Node.

2. Open Exercise 6.vi.

3. Save the VI as "Exercise 7.vi".

4. Go to the block diagram and delete the wire connecting the Simulate Signal VI to the Waveform Graph.

5. Place down a MathScript Node (Programming»Structures»MathScript Node).

6. Right-click on the left border of the MathScript Node and select **Add Input**. Name this input "In1" by typing while the input node is highlighted black.

7. Right-click on the right border of the MathScript Node and select **Add Output**. Name this output "Out".

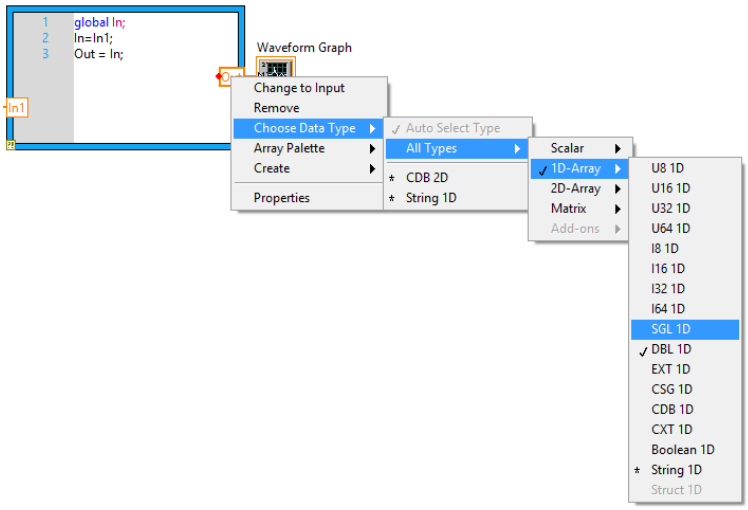
8. Convert the Dynamic Data Type output of the Simulate Signals VI to a 1D Array of Scalars to input to the MathScript Node. Place a Convert from Dynamic Data Express VI on the block diagram (**Express**»**Signal Manipulation**»**Convert from Dynamic Data**). By default, the VI is configured correctly so click "OK" in the configuration window.

9. Wire the "Sine" output of the Simulate Signal VI to the "Dynamic Data" input of the Convert from Dynamic Data VI.

10. Wire the "Array" output of the Convert from Dynamic Data VI to the "In1" node on the MathScript Node.

11. In order to use the data from the Simulate Signal VI in the Interactive MathScript Window it is necessary to declare a global variable and then place the value from "In1" in it. Inside the MathScript Node type "global In;" and on the next line "In=In1;". Initally we will just pass this variable straight through so type "Out = In;".

12. Right-click on the variable "Out" and select **Choose Data Type**»**All types**»**1D-Array**»**DBL 1D**. Output data types must be set manually on the MathScript Node.



13. Wire “Out” to the Waveform Graph.

14. Return to the front panel and increase the frequency to be between 50 and 100. Run the VI.

15. Open the Interactive MathScript Window (**Tools»MathScript Window…**).

16. In the MathScript Window, the Command Window can be used to enter in the command that you wish to compute. In the Command Window, type “global In” and press “Enter”. This will allow you to see the data passed to the variable “In” on the MathScript Node.

17. Note that all declared variables in the script along with their dimensions and type are listed on the “Variables” tab. To display the graphed data, click once on the variable **In** and change the drop down menu from “Numeric” to “Graph”.

18. Use the graph palette to zoom in on your data.

19. Multiply the data by a decreasing exponential function. Follow these steps:

a. Make a 100 element array of data that constitutes a ramp function going from 0.01 to 5 by typing “Array = [0.01:0.05:5];” in the Command Window and pressing Enter. What type of variable is “Array”?

b. Make an array containing a decreasing exponential. Type “Exp = 5*exp(-Array);” and press Enter.

c. Now multiply the Exp and In arrays element by element by typing “Out = In**.***Exp;” and pressing Enter. [Note the operator is “**.***” the matrix multiplication operator and not “***” the simple multiplication operator.]

d. Look at the graph of the variable “Out”.

20. Go to the History tab and use Ctrl-click to choose the 4 commands you just entered. Copy those commands using <Ctrl-C>.

21. On the Script tab, paste the commands into the Script Editor using <Ctrl-V>.

22. Save your script by clicking “Save” at the bottom of the window. Save it as “myscript.txt”

23. Close the MathScript Window.

24. Return to the block diagram of Exercise 4.2 – MathScript. Load the script you just made by right-clicking on the MathScript Node border and selecting **Import…** Navigate to myscript.txt, select it, and click “OK”.

25. Return to the front panel and run the VI.

26. Does the data look like you expect? Can you edit the program so that it responds to a change in frequency in real time?

27. Save and close your VI. Show your VI to the demonstrator.

Exercise 8

1. In this exercise, you will create a VI that uses what you have learned. Design a VI that does the following:

2. Acquire data from your device and graph it.

3. Filter that data using the Filter Express VI (**Functions»Express»Signal Analysis»Filter**). There should be a front panel control for a user configurable cut-off frequency.

4. Take a Fast Fourier Transform to get the frequency information from the filtered data and graph the result. Use the Spectral Measurements Express VI (**Functions»Express»Signal Analysis»Spectral**).

5. Find the dominant frequency of the filtered data using the Tone Measurements Express VI.

6. Compare that frequency to a user inputted limit. If the frequency is over that limit, light up an LED.

7. If you get stuck, ask the demonstrator for a hint.

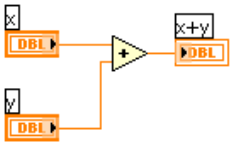
Exercise 9 – Creating a SubVI

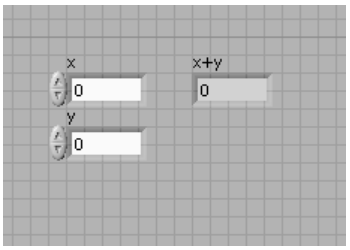
1. Create a subVI from a new VI, which adds two inputs and outputs the sum.

2. Open a new VI (**Ctrl+N**).

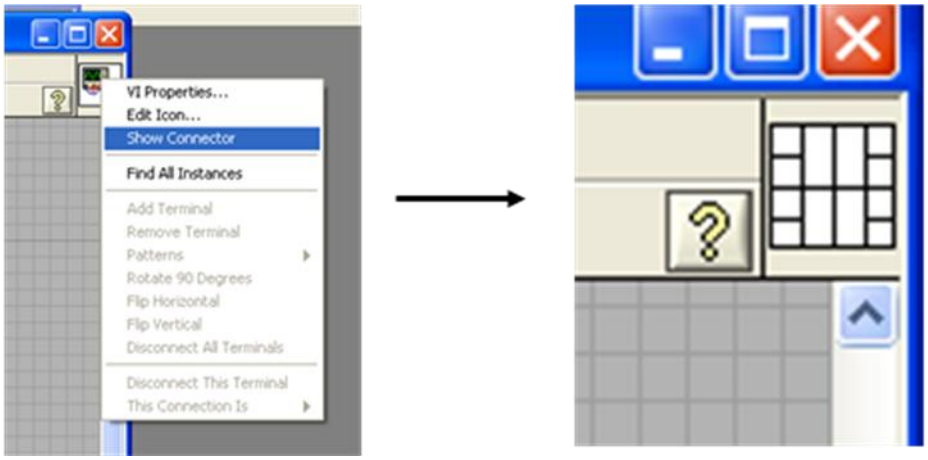
3. Place the Add function (**Programming » Numeric**) on the block diagram.

4. Create controls and indicators by right-clicking and selecting **Create » Control** or **Indictor**. The Block Diagram and Front Panel should look similar to the images below.



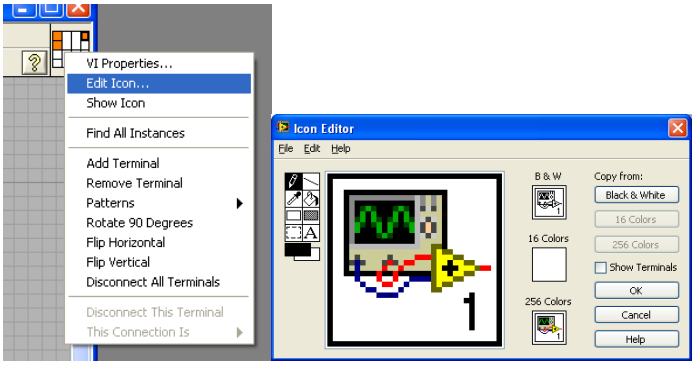


5. On the Front Panel right-click the Icon at the top right and select **Show Connector** to reveal the Connector Pane.



6. Assign icon terminals to the two controls and indicators by first left-clicking on a icon terminal and then clicking the desired control/indicator

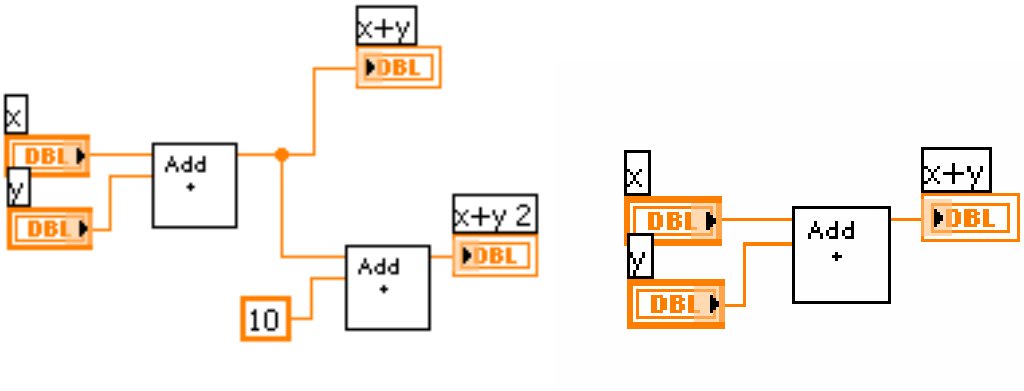
7. **Note:** General convention is to have controls as data inputs on the left side and indicators as outputs on the rights side of this icon.



8. Right-click on the Connector Panel and select **Edit Icon…**. This will bring up the Icon Editor.

9. Modify the graphics to more accurately represent the function of the SubVI, in this case Addition.

10. Save the SubVI. It can now be used in any other VI to perform any function, in this case adding two numbers.

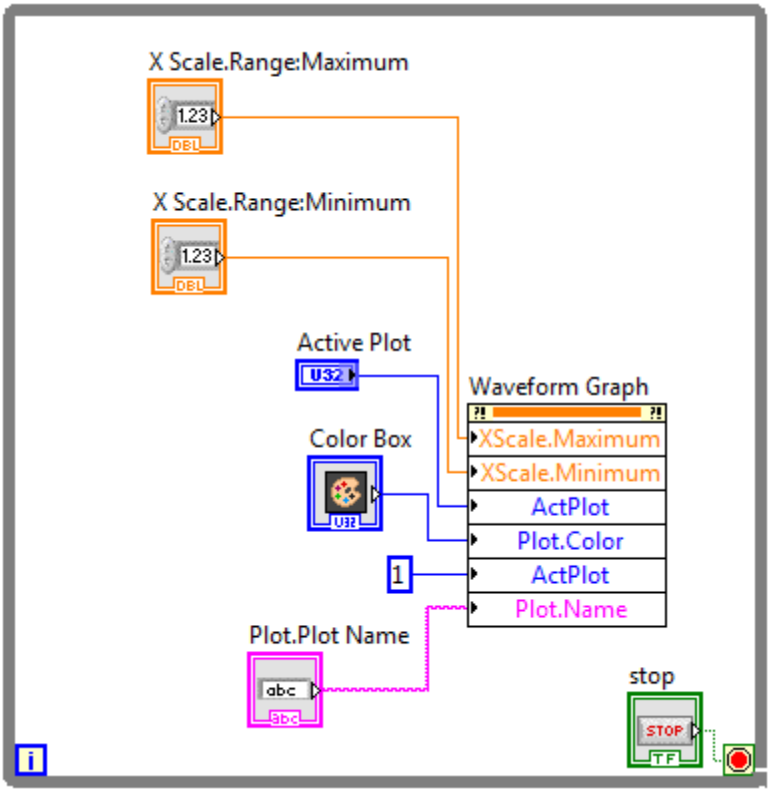


Advanced Labview Concepts

We will now explore some more involved aspects of working with Labview. Many of these will be important if you want your program to run on multiple machines or be used by other people. The details given here are only an overview, where applicable there is a link to further information.

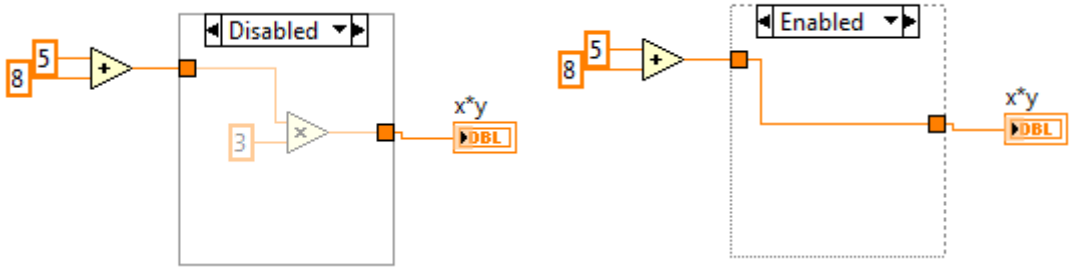
Property nodes

Many of the variables normally controlled through the settings dialog boxes can also be controlled programmatically using *property nodes.* Visual elements like colours and labels can be changed but it is also possible to query front panel objects in ways not normally possible.



To create a property node, right click on the object you wish to control then **Create – Property Node**.

Diagram Disable



This tool (found in the structures panel) allows you to temporarily disable areas of code. This

is very useful when trying to track down bugs.

Strings

Strings are pieces of text data which can be manipulated just like any other data type. A primary use is for file names where certain tasks are often useful:

String Length: Returns in length the number of characters (bytes) in string.

Array To Spreadsheet String Converts an array of any dimension to a table in string form, containing tabs separating column elements, a platform-dependent EOL character separating rows, and, for arrays of three or more dimensions, headers separating pages.

Concatenate Strings Concatenates input strings and 1D arrays of strings into a single output string. For array inputs, this function concatenates each element of the array.

Match Pattern Searches for regular expression in string beginning at offset, and if it finds a match, splits string into three substrings. A regular expression requires a specific combination of characters for pattern matching.

Dialog Boxes

The “Dialog and User Interface” group features a number of VIs which allow the user to make decisions. Popup boxes can be useful when setting up the VI for the first time, but use them with care – too many can be very frustrating!

Paths

Paths are a specific type of string which carry file location data. You can ask a user to select a file location using a dialog box, or you can use a number of preset file locations such as the correct “My Documents” folder, or the location where the VI itself is saved. There are type categories of file path; relative and absolute. A relative path describes the location of a file or directory relative to an arbitrary location in the file system. An absolute path describes the location of a file or directory starting from the top level of the file system Using relative paths in VIs avoids having to rework the paths if you build an application or run the VI on a different computer.

Digital vs analog, TTL, sampling rates and Nyquist theory

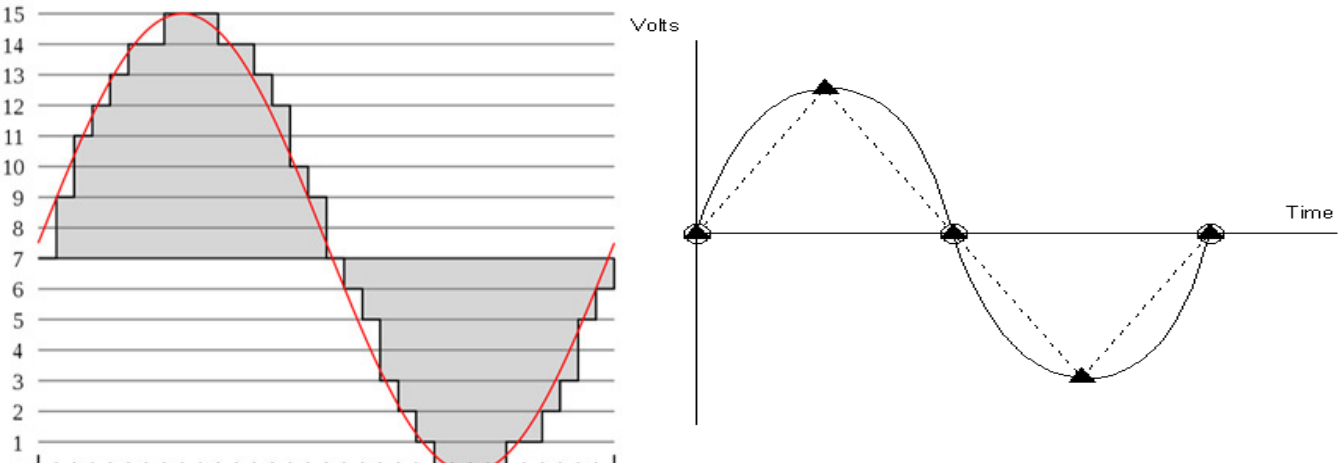
We will be using a NI USB-6211 - 16-Bit, 250 kS/s M Series Multifunction DAQ. But what does this mean? The Data Aquistion box (DAQ) has 4 key features:

- Up to 32 analog inputs at 16 bits, up to 400 kS/s (250 kS/s scanning)
- Up to 2 analog outputs at 16 bits
- Up to 32 TTL/CMOS digital I/O lines
- Two 32-bit, 80 MHz counter/timers

Analog to digital conversion

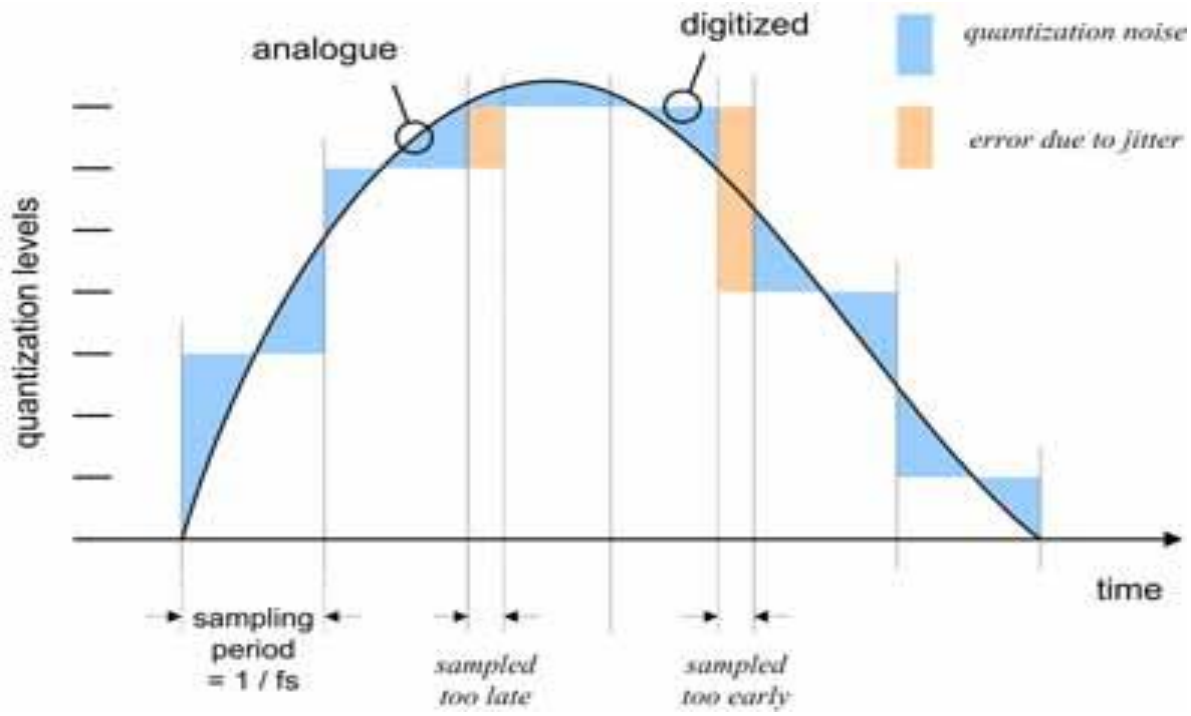
The DAQ box can measure analog inputs, however in order to process this using a digital system (a computer) this analog signal must be digitised. This is where the "16 bits, up to 400 kS/s " is important.

To digitise an analog signal is is broken down into a number of discrete values (digitzation levels or bins). The number of these levels defines the resolution in terms of the measurement value, in this case 16 bit. This means there are 2^{16} of these levels (65,536). The next important value is how quickly the input value is quantised, this determines the resolution in time. The NI USB-6211 can make 400,000 measurements a second.



$$Code\,width = \frac{Device\ Input\ Range}{2^{Resolution}} = \frac{10V}{2^{16}} = 305\mu V$$

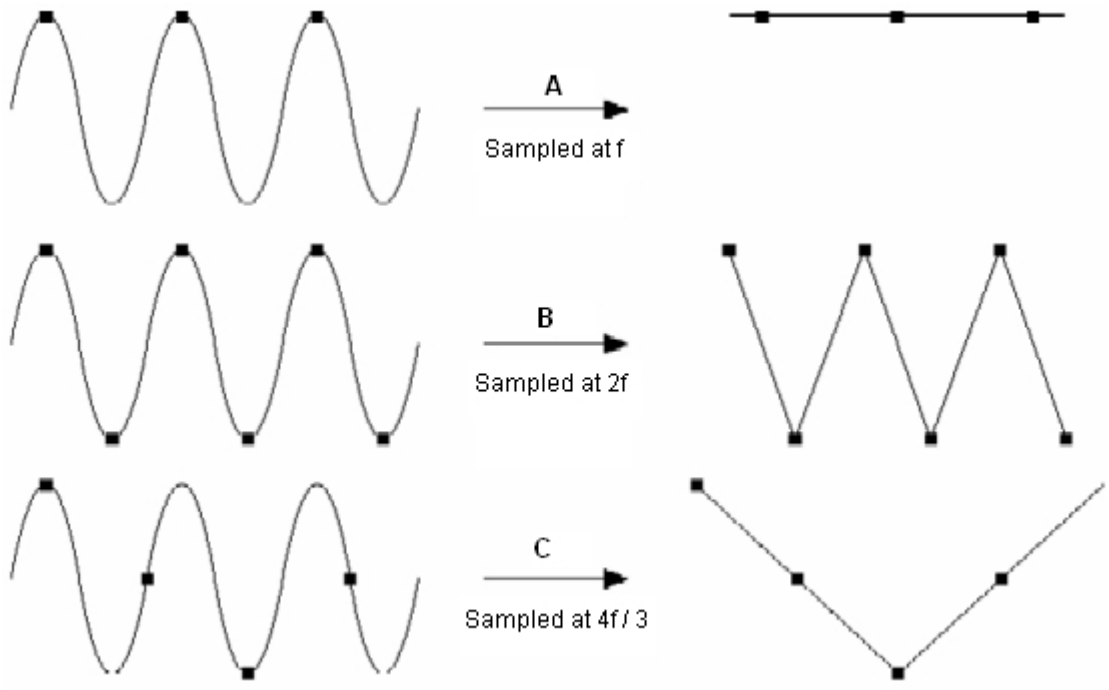
The finite resolution of the measurements means there is an error associated with the digitisation, a state of the art A/D converter will run at 2.5 GS/s.



Nyquist Theorm

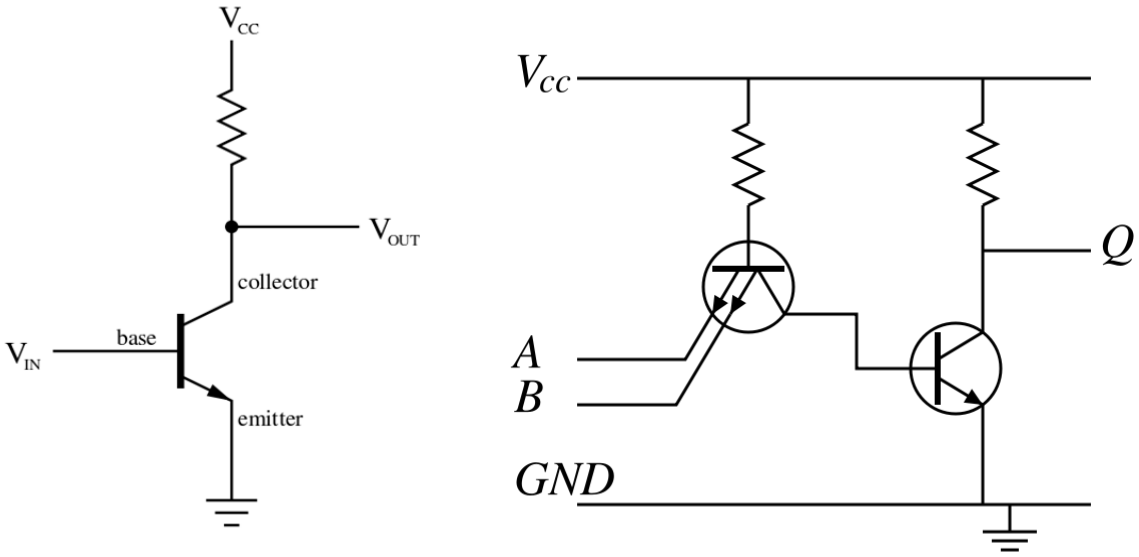
Nyquist Theorem: Sampling rate (f s) > 2 * highest frequency component (of interest) in the measured signal

The Nyquist theorem states that a signal must be sampled at a rate greater than twice the highest frequency component of interest in the signal to capture the highest frequency component of interest; otherwise, the high-frequency content will alias at a frequency inside the spectrum of interest (pass-band).



TTL and CMOS

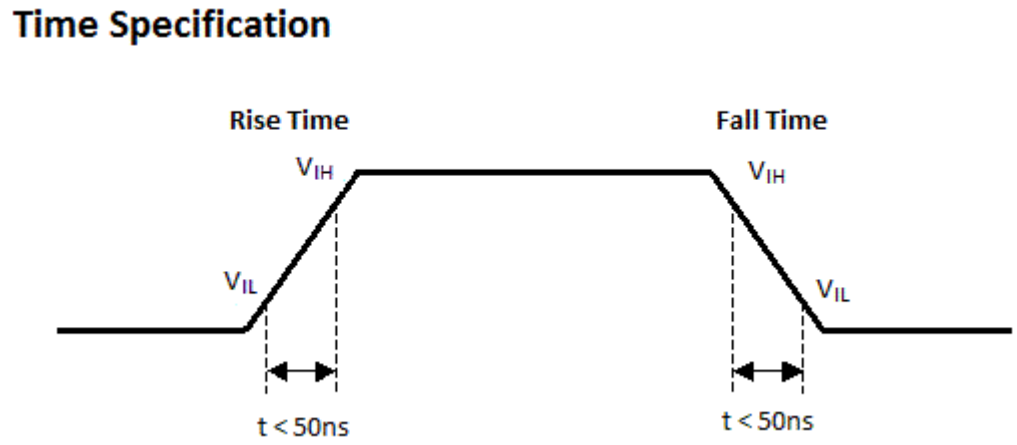
Transistor–transistor logic (TTL) is a class of digital circuits built from transistors and resistors.

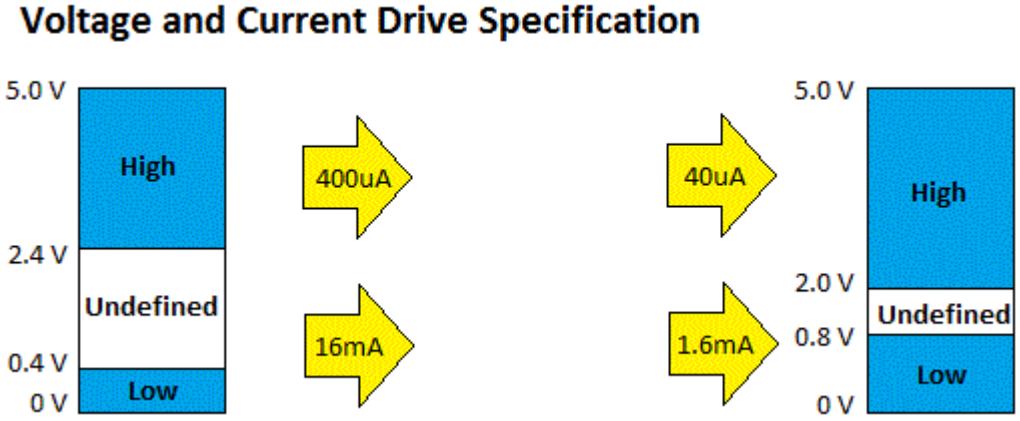


The transistor-transistor-logic (TTL) family was developed in the use of transistor switches for logical operations and defines the binary values as:

0 V to 0.8 V = logic 0
2 V to 5 V = logic 1

The maximum rise/fall time is 50 ns between VL and VH.





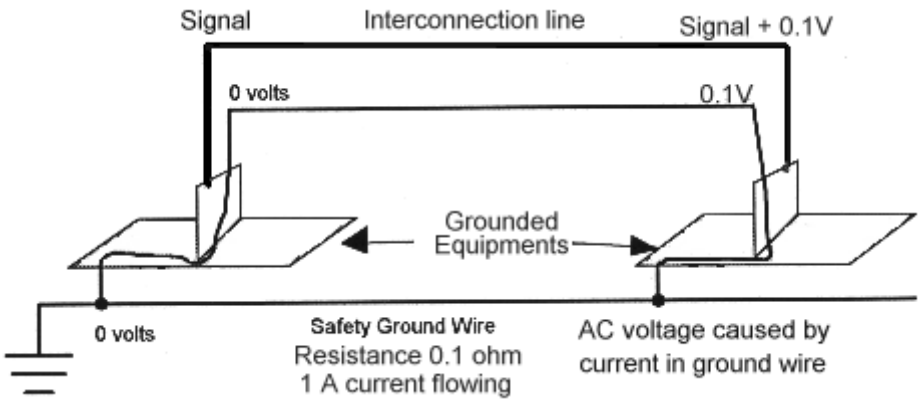
Ground and ground loops
http://www.ni.com/white-paper/3394/en/

At a simple level, ground (or earth) is the reference zero point from which voltages can be measured. This common zero makes it possible to measure the differential voltage at different sites. If a circuit is not connected to ground then it is referred to as “floating”.

Power supplies and any equipment connected through a main supply will typically be grounded to earth. Self contained circuits like multimeters or circuits running from a battery will typically not reference earth and therefore are floating.

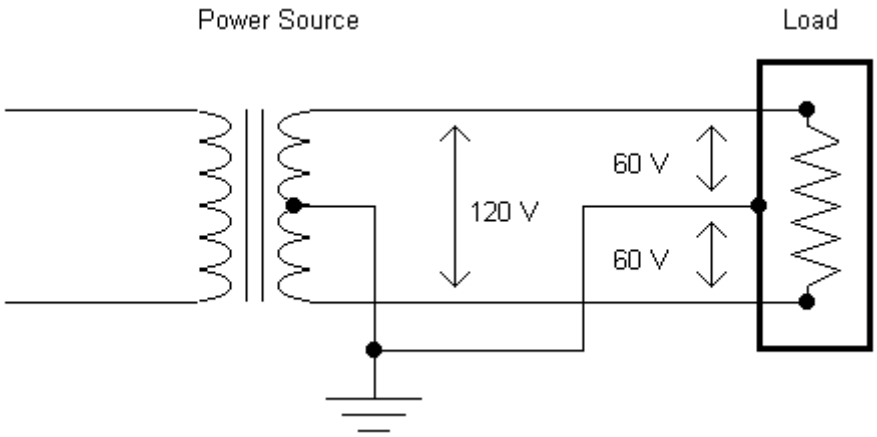
While ground can often be thought of as a perfect sink for charge, in that it never has a voltage build up on it, in reality for very sensitive measurements there can be problems with using a normal grounding.

The grounds of two independently grounded signal sources generally will not be at the same potential. The difference in ground potential between two instruments connected to the same building ground system is typically 10mV to 200mV, or even more. The difference can be higher if power distribution circuits are not properly connected.



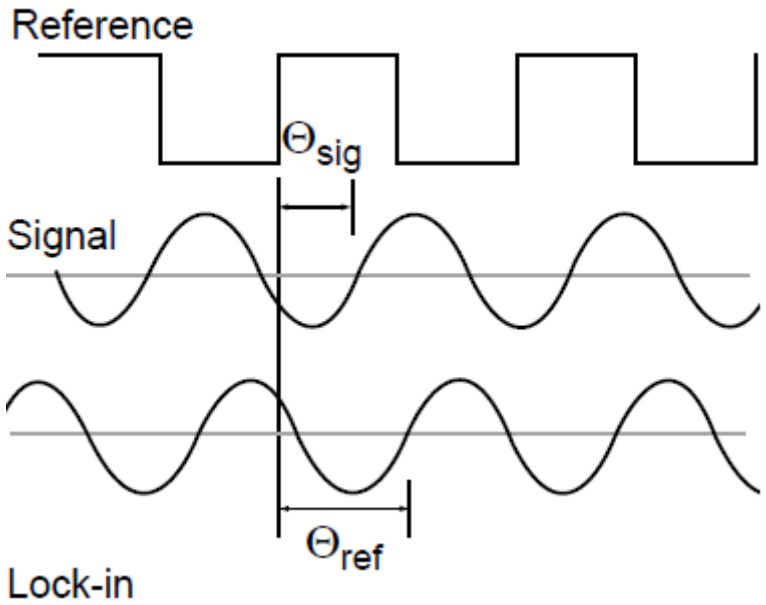
Because there is voltage difference between the equipment, the signal in the interconnection wire sees that difference added to signal.

Additionally if a large number of devices are sharing an earth within a building, it is likely the potential will fluctuate. Therefore it is often preferable to have a “clean” or isolated earth. Using an isolation transformer allows you to have a second circuit with its own, very good connection to earth such that you can be confident it is constant (you can also bias this reference to an arbitrary value if required). Using a transformer to pass power from the main supply prevents high frequency noise.



Lock-in amplifiers
http://www.thinksrs.com/downloads/PDFs/ApplicationNotes/AboutLIAs.pdf

Lock-in amplifiers allow you to measure very small signals buried in large amounts (several orders of magnitude larger) of background signal noise. By exciting the experiment at a known frequency, the lock-in can act as a very narrow band pass filter which then extracts only the signal you require. In essence, a lock-in amplifier takes the input signal, multiplies it by the reference signal (either provided from the internal oscillator or an external source), and integrates it over a specified time, usually on the order of milliseconds to a few seconds. The resulting signal is a DC signal, where the contribution from any signal that is not at the same frequency as the reference signal is attenuated close to zero.



Firstly, a signal is created by exciting the experiment using an external sine wave:

$$\text{Signal} = V_{sig}\sin(\omega_r t + \theta_{sig})$$

The lock-in amplifier then generates its own internal reference, phase locked to an external reference:

$$\text{Reference} = V_L \sin(\omega_L t + \theta_{ref})$$

The lock-in amplifies the signal and then multiplies it by the lock-in reference using a phase-sensitive detector or multiplier. The output of the PSD is simply the product of two sine waves.

$$V_{PSD} = V_{sig}V_L \sin(\omega_r t + \theta_{sig})\sin(\omega_L t + \theta_{ref})$$

$$= ½ V_{sig}V_L \cos([\omega_r-\omega_L]t + \theta_{sig} - \theta_{ref}) – ½ V_{sig}V_L \cos([\omega_r+\omega_L]t + \theta_{sig} - \theta_{ref})$$

The PSD output is two AC signals, one at the difference frequency ($\omega_r - \omega_L$) and the other at the sum frequency ($\omega_r + \omega_L$). If the PSD output is passed through a low pass filter, the AC signals are removed. If ω_r equals ω_L, the difference frequency component will be a DC signal. In this case, the filtered PSD output will be:

$$V_{psd} = ½ V_{sig}V_L \cos(\theta_{sig} - \theta_{ref})$$

This is a very nice DC signal proportional to the signal amplitude.

Note the later sections of each exercise are challenging and open ended, do not be dispirited if your program does not work very well the important thing is to show you can program in Labview and understand (and explain in your report) the limitations of your program

Assessed Exercise A – Pressure sensor

A pirani is a widely used tool to measure vacuum pressures. The pirani consists of a metal wire, the resistance of which varies with pressure. This variation is calibrated and gives a voltage output in relation to the pressure as shown in the figure on the next page.

In this exercise you should find a suitable model to convert the voltage to the pressure and use this to produce a program that can display an incoming voltage into a pressure. You will be provided with a VI that will simulate a voltage output. How should this pressure best be presented? Can you include a high/low pressure alarm? What other features might be useful?

Assessed Exercise B - Sound Analysis

Write a program that listens for a particular sound (breaking glass is not safe in the lab but what about a particular object hitting the bench) and then raises an alarm. Can you arrange for the program to let the sample sound be changed programmatically? Can this program be extended to listen to someone speaking a specific word and, by analyzing the tonal content of their voice, recognize them. Ideally your program will be able to tell the difference between two or three different people and programmatically learn new voices. Can you extend your program to perform different actions in response to different spoken commands.

Assessed Exercise C – Vision tracking

Automatic person tracking software is a well developed technology which allows security cameras to track a person as they move across the field of view. However they involve complex algorithms and do not work well in low light.

The aim of this project is to use a light source as an indicator of a person and use this to track their movement.

You have been provided with the following items:

A stepper motor and controller

A webcam

3 light sensors

A NI DAQ box

By using these devices it should be possible to scan across a room, detect an intruder and acquire an image of them. This image should then be saved for future reference.

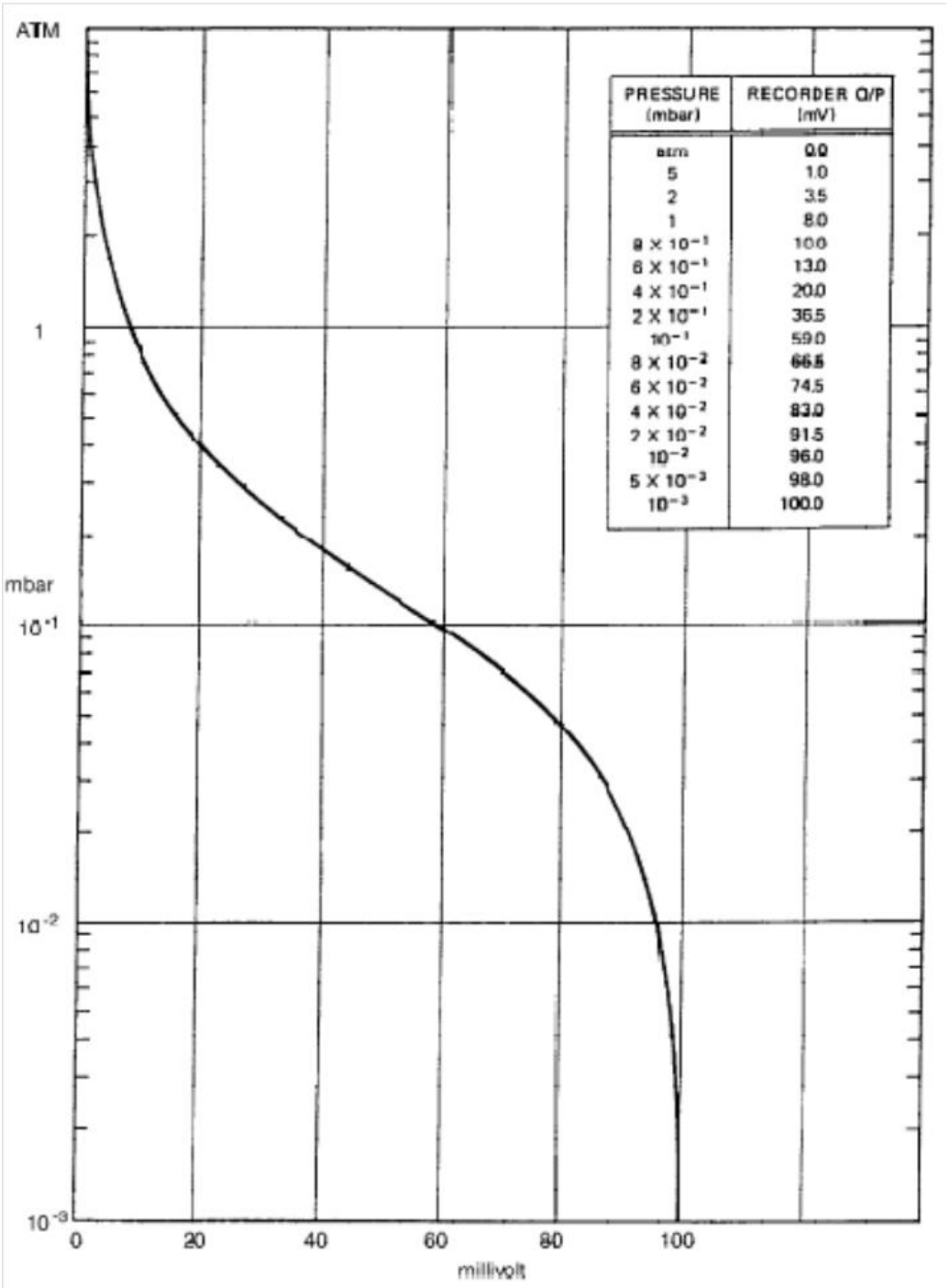


Figure 1 - Pirani 501 recorder output graph (mbar)

Stepper Motor

The stepper motor is of the unipolar type and is provided with a drive board. You do not need to worry about the exact mode of operation of the motor, simply plug it into the drive board. The board needs a 12V power supply, with which you are provided. The white striped wire is +ve and the other –ve. At the back of this handout is the data sheet for the control board you can use to wire it correctly.

The drive board has a number of TTL level inputs, you are probably most interested in those that control direction and step, the level of the direction input controls the direction of the motor and every pulse sent to the step input will cause the stepper motor to move one step. Using the counter on the DAQ unit you will be able to send the stepper motor a series of pulses that will cause it to move. The stepper motor moves 1.8 degree per pulse, you can set the driver board to "half-step" mode which will cause the motor to move half that per pulse. You can use the "Driver EN" input to shut down the driver when the motor is not needed to save power.

Remember that you will need to connect the digital ground on the DAQ board to the ground on the driver board for the signals to be read.

Light Sensors

The Light Sensors have 3 wires, +5V (Red),0V (black) and output. Configure the analog input for differential mode (where each channel measures the voltage between two inputs). You will need to connect the light sensor to the +5V output of the DAQ board and to the "Digital Ground". The output needs to go to the + terminal of one of the analog inputs. Remember that you will need to connect the –ve input to the digital ground terminal in order to measure the output from the light sensor.

The data sheet for the light sensor is at the back of this handout. It has a non-linear response closely matched to that of the human eye and will output about 4.2V in bright light. The Light Sensor has been pre-wired for you with a 5.1 kΩ resistor.

USB Camera

Using the LabVIEW Vision Development module you will be able to simply build a VI that takes a picture from the camera. The system is much more powerful than this and you may wish to spend some time exploring the features of the software. If you get stuck ask a demonstrator.

DAQ Pinout

The counter outputs by default on PFI4. Do not address the output lines as a port since this will prevent you using PFI4 as a counter output, address the output lines individual in labview.

Bus-Powered M Series Multifunction DAQ for USB - 16-Bit, up to 400 kS/s, up to 32 Analog Inputs, Isolation

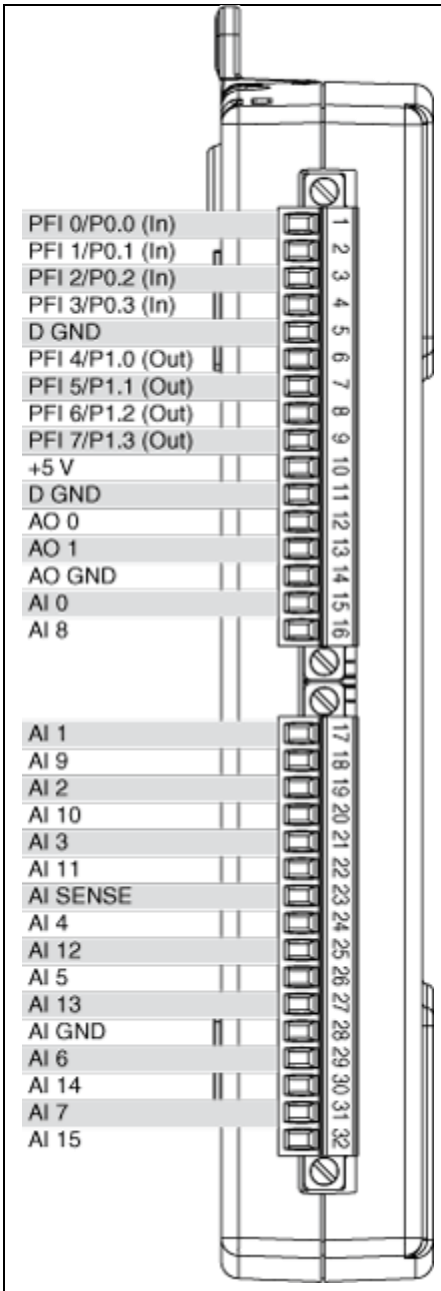
- Up to 32 analog inputs at 16 bits, up to 400 kS/s (250 kS/s scanning)
- Up to 2 analog outputs at 16 bits
- Up to 32 TTL/CMOS digital I/O lines
- Two 32-bit, 80 MHz counter/timers
- NI-PGIA 2 and NI-MCal calibration technology for improved measurement accuracy
- NI signal streaming for 4 high-speed data streams on USB
- Bus-powered design
- Available with CAT I isolation

Overview

With recent bandwidth improvements and new innovations from National Instruments, USB has evolved into a core bus of choice for measurement and automation applications. NI M Series devices for USB deliver high-performance data acquisition in an easy-to-use and portable form factor through USB ports on laptop computers and other portable computing platforms. National Instruments designed the new and innovative patent-pending NI signal streaming technology that enables sustained bidirectional high-speed data streams on USB. The new technology, combined with advanced external synchronization and isolation, helps engineers and scientists achieve high-performance applications on USB.

NI M Series bus-powered multifunction data acquisition (DAQ) devices for USB are optimized for superior accuracy in a small form factor. They provide an onboard NI-PGIA 2 amplifier designed for fast settling times at high scanning rates, ensuring 16-bit accuracy even when measuring all available channels at maximum speed.

All bus-powered devices have a minimum of 16 analog inputs, digital triggering, and two counter/timers. USB M Series devices are ideal for test, control, and design applications including portable data logging, field monitoring, embedded OEM, in-vehicle data acquisition and academic.



Analog Input

Number of channels	8 differential or 16 single ended
Maximum Sampling rate	250 kS/s single channel, 250 kS/s multichannel (aggregate)
Timing resolution	50 ns
Input coupling	DC
Input range	±10 V, ±5 V, ±1 V, ±0.2 V ±10.4 V of AI GND

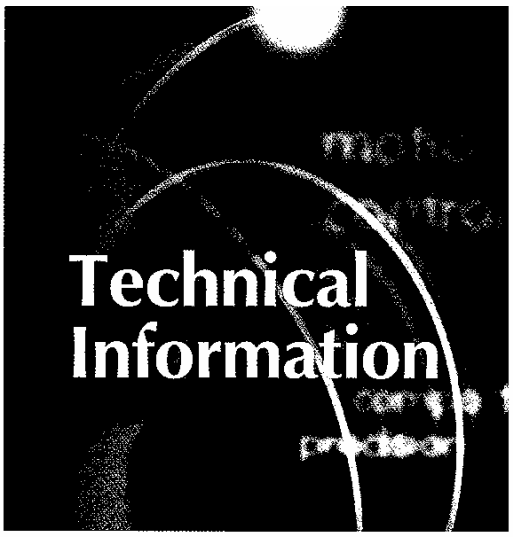
Analog Output

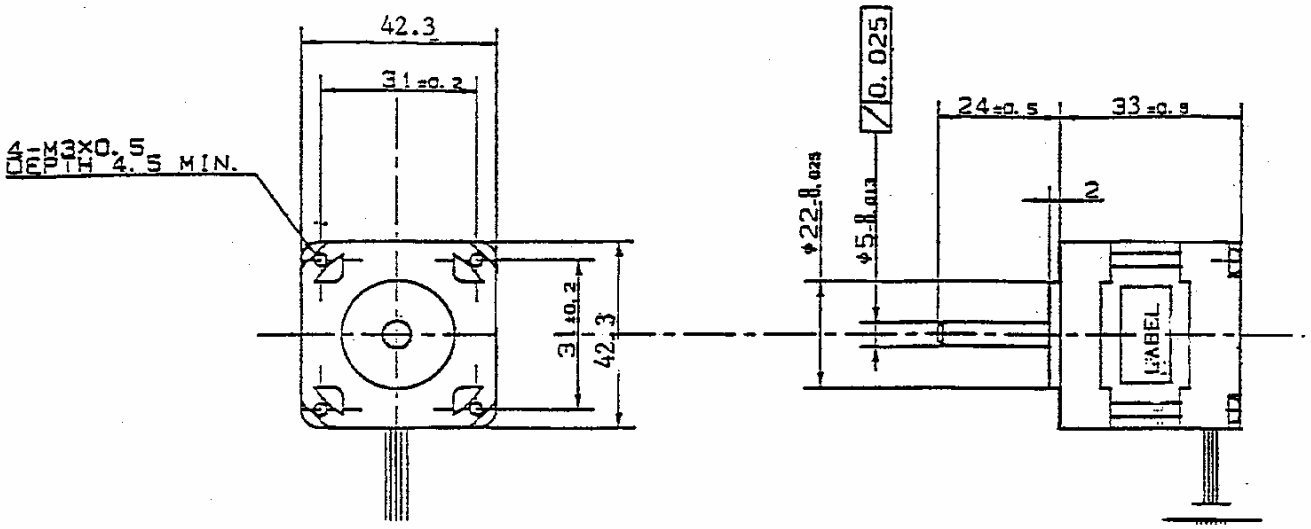
Number of channels	2
DAC resolution	16 bits
Maximum update rate	
1 channel	250 kS/s
2 channels	250 kS/s per channel
Timing accuracy	50 ppm of sample rate
Timing resolution	50 ns
Output range	±10 V
Output coupling	DC
Output impedance	0.2 Ω
Output current drive	±2 mA
Power-on state	±20 mV
Power-on glitch	±1 V for 200 ms
Output FIFO size	8,191 samples shared among channels used
Data transfers	USB Signal Stream, programmed I/O

General-Purpose Counter/Timers

Number of counter/timers	2
Resolution	32 bits
Counter measurements	Edge counting, pulse, semi-period, period, two-edge separation
Internal base clocks	80 MHz, 20 MHz, 0.1 MHz
External base clock frequency	0 MHz to 20 MHz
Base clock accuracy	50 ppm

Stepping Motor
Type Y129





Specification		Y129
Step angle		1.8°
Step angle accuracy	%	5
Phase current	A	0.16
Phase resistance	Ω	75
Phase inductance	mH	60
Holding torque unipolar	Ncm	9
Holding torque bipolar	Ncm	-
Detent torque	Ncm	1.5
Rotor inertia	g cm 2	28
Mass	Kg	0.22
Insulation Class		B
Shaft configuration		Single
Number of leads		6

All dimensions in mm

Issue 01 01/03

ASTROSYN
ADVANCING THE DRIVE TOWARDS MOTOR EFFICIENCY

Saia-Burgess Dresden AG
Karl-Liebknecht-Strasse 6 l D-01257 Dresden l Germany
T +49 (0)351 207 860 l F +49 (0)351 207 863 61 l www.Saia-burgess.com

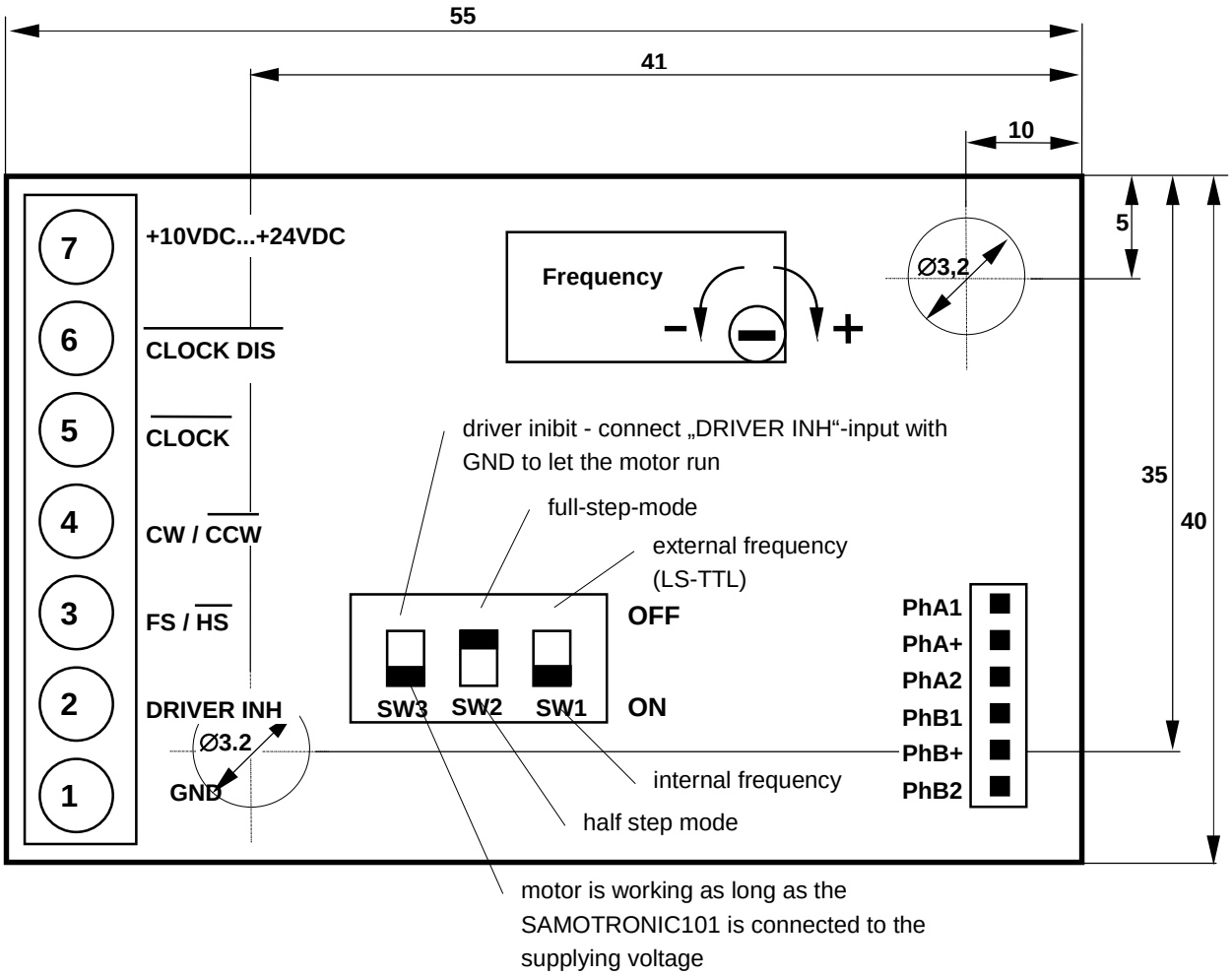
Smart solutions for comfort and safety

SAMOTRONIC101 (4 636 6608 0)

Short Reference Unipolar Stepper Motor Driver

1. Overview

All dimensions in mm. Tolerance of hole diameter ±0.1mm, of all other dimensions ±0.5mm.



- CW / FS / DRIVER INH... functions are logic HIGH active (LS-TTL-level >2 V to 5 V); they apply automatically when input is open (depending on the switch position)
- CLOCK DIS ... functions are logic LOW active (LS-TTL-level 0 V to <0.8 V); no HIGH signal allowed (leave open instead)
- CCW / HS ... functions are logic LOW active (LS-TTL-level 0 V to <0.8 V)
- CLOCK ... one step is generated for each negative edge of the clock signal (clock signal LS-TTL-level)

Take care on the input levels (LS-TTL-signals). For LOW a real ground connection is required. If an input is left open it is detected as HIGH. Input signals >5V may destroy the SAMOTRONIC101.

Saia-Burgess Dresden AG
Karl-Liebknecht-Strasse 6 I D-01257 Dresden I Germany
T +49 (0)351 207 860 I F +49 (0)351 207 863 61 I www.Saia-burgess.com

saia-burgess
Smart solutions for comfort and safety

- field of operation: unipolar stepper motors; I ≤ 350 mA/phase
- power supply : 10...24 (+10%) VDC
- connectors: plug connector for the motor
  screw connector for supplying voltage and external control signals

Functions

Logic level: LS-TTL LOW 0V to <0.8V
HIGH >2V to 5V

function	**external connector**	**setting on the PCB**	**variable via screw connector**
step frequency		S1 = On, adjustable via on-board potentiometer f_S = 50(± 20%)...390 Hz(±10%) (accuracy ±15%)	use port CLK to access the internal generated clock signal
	CLOCK	S1 = Off	connect to external clock (TTL) via shielded cable
full step mode		S2 = Off	
half step mode		S2 = On	
switching between full/half step mode	FS/HS	S2 = Off	HIGH or not connected ⇒ full step LOW ⇒ half step
stop signal		S3 = On, motor is working with step frequency as long as supplying voltage appears	
	DRIVER INH	S3 = Off	LOW ⇒ motor is working with step frequency HIGH or not connected ⇒ motor stands still, phases switched off
	CLK DIS		LOW ⇒ clock stopped, motor still under current HIGH is not allowed!
direction	CW/CCW		TTL-signal 0 V – CCW HIGH or not connected – CW

The single functions are fixable with help of the DIP switches or controllable with external signals. With fixed functions it is only necessary to connect the SAMOTRONIC101 to the supplying voltage and to change the direction if necessary.

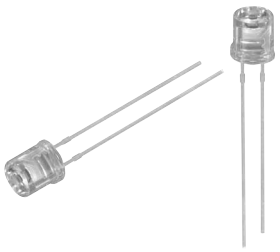
Mounting: The unit must be installed in a case so that the user is not able to access it while working. Otherwise and for maintenance purpose the unit must be used in an antistatic environment and the user must be discharged with standard ESD-protection equipment. Take care to leave minimum distance to holes and gapes in the case to prevent electrostatic discharges. A typical value for 8kV (EN50082-1/2) is 8mm (DIN-VDE0110-1), depending on environment, humidity and temperature conditions.

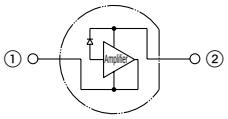
Cables: Max. cable length for all control inputs (screw terminal port 2-6) is 3m. At the clock input (screw terminal port 5) shielded wires must be used.

Panasonic
ideas for life

Cadmium-free sensor with spectral response The Through-hole Type for easy implementation as a CdS cell replacement

Light Sensor
NaPiCa
"Through-hole Type"

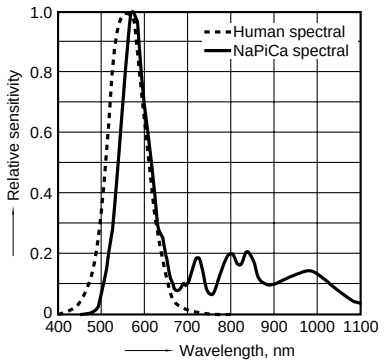




① Anode
② Cathode

FEATURES

1. Built-in optical filter for spectral response similar to that of the human eye.
Peak sensitivity wavelength is 580 nm



2. Photocurrent is proportional to illumination. (linear output)
High photocurrent is achieved by built-in photocurrent amp.
I_L = 260 µA (typical)
Ev = 100 lx (florescent light)
3. Uses environmentally friendly silicon chips.
4. Lead-free.
5. Operates on 1.5 to 6 V DC, which is suitable for battery operation.
6. Same through-hole shape as CdS cell.

TYPICAL APPLICATIONS

1. **Brightness detection for LCD backlight control for LCD devices (LCD TVs, car navigation systems, etc.)**
2. **Brightness detection for circuits in household lighting, crime prevention lighting, and automatic lighting for bicycle.**
3. **Brightness detection for wall clocks (radio clocks).**
4. **Brightness detection for surveillance cameras (night switching).**

TYPES

Photocurrent	Part No.	Packing quantity
260 µA*	AMS302T	2,000pcs.

Note: Ev = 100 lx (Fluorescent lamp is used as light source)
Tape package is the standard packing style.

RATINGS

1. Absolute maximum ratings (Ambient temperature: 25°C 77°F)

Item	Symbol	AMS302	Remarks
Reverse voltage	V_R	–0.5 to 8 V	—
Photocurrent	I_L	5 mA	—
Power dissipation	P	40 mA	—
Operating temperature	T_{opr}	–30 to +85°C –22 to +185°F	Non-condensing at low temperatures
Storage temperature	T_{stg}	–40 to +100°C –40 to +176°F	Non-condensing at low temperatures

2. Recommended operating condition

Item		Symbol	AMS302	Remarks
Reverse voltage	Minimum	V_R	1.5 V	—
	Maximum		6 V	

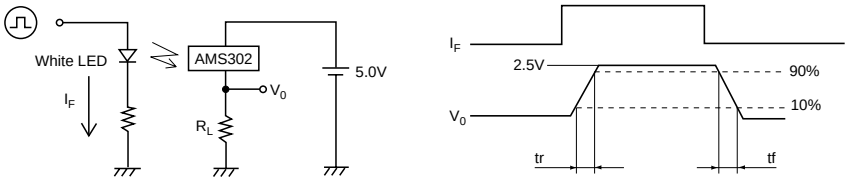
All Rights Reserved © COPYRIGHT Matsushita Electric Works, Ltd.

Light Sensor (AMS3)

3. Electrical characteristics (Ambient temperature: 25°C 77°F)

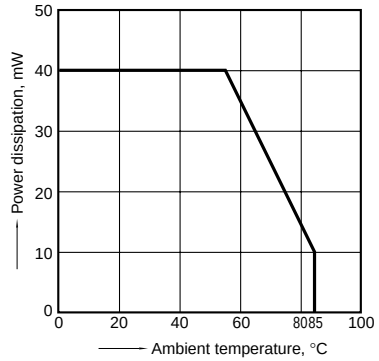
Item			Symbol	AMS302	Condition
Peak sensitivity wavelength		—	λ_p	580 nm	—
Photocurrent 1		Minimum	I_{L1}	9.1µA	V_R = 5 V, E_V = 5 lx*1
		Typical		13µA	
		Maximum		16.9µA	
Photocurrent 2		Minimum	I_{L2}	182µA	V_R = 5 V, E_V = 100 lx*1
		Typical		260µA	
		Maximum		338µA	
Photocurrent 3		Typical	I_{L3}	500µA	V_R = 5 V, E_V = 100 lx*2
Dark current		Maximum	I_D	0.3µA	V_R = 5 V
Switching time	Rise time	Typical	t_r	8.5 ms	V_R = 2.5 V, V_O = 2.5 V, R_L = 5kΩ
	Fall time	Typical	t_f	8.5 ms	

*1 Fluorescent lamp is used as light source. (Substituted with white LED for testing before shipping.)
*2 CIE standard illuminant 'A' is used as light source.
* Measuring method for switching time.

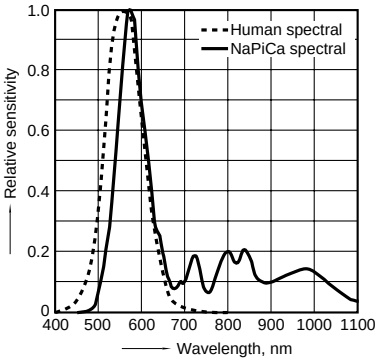


REFERENCE DATA

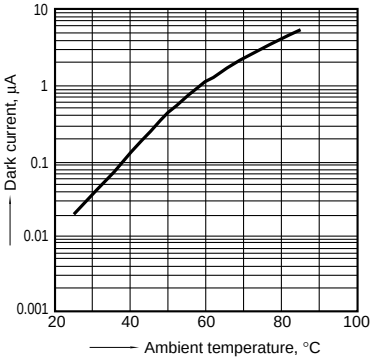
1. Power dissipation vs. ambient temperature characteristics



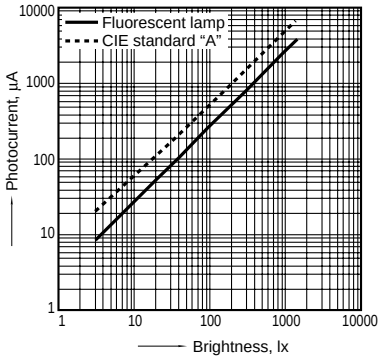
2. Relative sensitivity vs. wavelength characteristics
Ambient temperature: 25°C 77°F



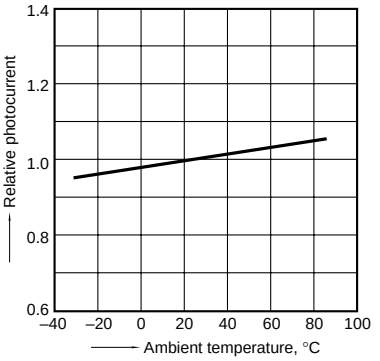
3. Dark current vs. ambient temperature characteristics
Reverse voltage: 5V



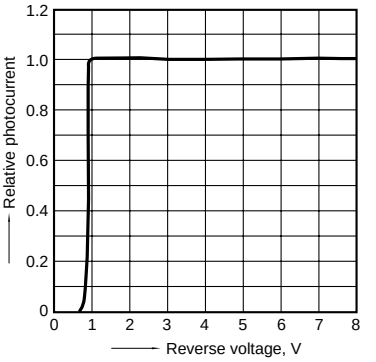
4. Photocurrent vs. brightness characteristics
Light source: Fluorescent lamp, CIE standard "A"
Reverse voltage: 5V, Ambient temperature: 25°C 77°F



5. Relative photocurrent vs. ambient temperature characteristics
Light source: Fluorescent lamp, Brightness: 100 lx
Reverse voltage: 5V



6. Relative photocurrent vs. reverse voltage characteristics
Light source: Fluorescent lamp, Brightness: 100 lx
Ambient temperature: 25°C 77°F



All Rights Reserved © COPYRIGHT Matsushita Electric Works, Ltd.